RL-TR-94-98
Final Technical Report
July 1994

SOFTWARE TECHNIQUES FOR BALANCING COMPUTATION & COMMUNICATION IN PARALLEL SYSTEMS

Purdue University

John K. Antonio

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-94-98 has been reviewed and is approved for publication.

APPROVED: Richard C. Metzger

RICHARD C. METZGER
Project Engineer

FOR THE COMMANDER: John A. Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1994	Final Aug 92 - Dec 93

4. TITLE AND SUBTITLE
SOFTWARE TECHNIQUES FOR BALANCING COMPUTATION & COMMUNICATION IN PARALLEL SYSTEMS

5. FUNDING NUMBERS
C - F30602-92-C-0108
PE - 62702F
PR - 5581
TA - 18
WU - P5

6. AUTHOR(S)
John K. Antonio

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Purdue University
School of Electrical Engineering
1285 Electrical Engineering Building
West Lafayette IN 49707-1285

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3CB)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-94-98

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Richard C. Metzger/C3CB/(315) 330-7650

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
The work presented in this report is the result of collaboration between Rome Laboratory and Purdue University. The work is divided into two main parts. The first part describes a new mapping technique developed under this effort called the hypersphere mapper, for the hypercube embedding problem. Solving the hypercube embedding problem involves finding the best mapping (i.e., assignment) of software tasks onto the processing elements of a parallel computer having a hypercube interconnection network. The second part of the work describes a new software engineering tool called OMARS (Optimal Mapping Alternate Routing System). OMARS is an interactive tool that aids the software engineer in deciding how to best map multiple software tasks onto the processors of message passing parallel architectures. Hypersphere mapper is one of several mapping algorithms that is integrated into OMARS.

14. SUBJECT TERMS
Static Task Allocation, Hypercube, Mapping, Scheduling, Objective Function

15. NUMBER OF PAGES
56

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Final Report

for

Software Techniques for Balancing Computation & Communication in Parallel Systems

Contents

Executive Summary

The work presented in this report is the result of collaboration between Rome Laboratory and Purdue University. The work is divided into two main parts. The first part describes a new mapping technique developed under this effort, called hypersphere mapper, for the hypercube embedding problem. Solving the hypercube embedding problem involves finding the best mapping (i.e., assignment) of multiple software tasks onto the processing elements of a parallel computer having a hypercube interconnection network. The second part of the work describes a new software engineering tool called OMARS (Optimal Mapping Alternate Routing System). OMARS is an interactive tool that aids the software engineer in deciding how to best map multiple software tasks onto the processors of message passing parallel architectures. Hypersphere mapper is one of several mapping algorithms that is integrated into OMARS.

1 Overview and Rationale of the Effort

1.1 The Mapping Problem

The specific motivation for investigating the mapping problem is rooted in a need to aid the parallel software engineer in developing and/or managing extremely complex, large-scale, and dynamic parallel software. Of particular concern is the development, management, and effective execution of large software applications that are truly functionally parallel in nature. Although the concept of functional parallelism has existed since the beginning of the parallel processing era, most of the existing parallel applications to date are based on the data parallel programming model (and not the functionally parallel programming model). It is believed that the next generation C3I systems will comprise large-scale and functionally parallel applications, some of which may only be well-suited for execution under the MIMD (multiple instruction stream - multiple data stream) programming model, which can support multiple independent threads of control.

A survey of the literature was performed in the area of automated techniques for mapping software tasks onto MIMD architectures. From the survey, a number of proposed techniques for architectures having a wide range of structures (e.g., hypercube-based versus mesh-based architectures) were found. One of the most popular mapping formulations, from those found in the literature, is the so-called hypercube embedding problem, which involves mapping tasks onto processors interconnected with the hypercube interconnection network. Because of the popularity of the hypercube embedding problem within the technical literature and because we had access to a hypercube-based machine (the nCUBE 2 located at Purdue University), we decided to further investigate the hypercube embedding problem. This investigation resulted in the development of a new hypercube embedding heuristic called hypersphere mapper, which is the topic of Section 2 of this report. Also, a technical paper (co-authored with Richard C. Metzger of Rome Laboratory) entitled "Hypersphere Mapper: A Nonlinear Programming Approach to the Hypercube Embedding Problem," was published in the *Proceedings of the 7th International Parallel Processing Symposium*, April 1993, pp. 538–547 and in the *Journal of Parallel and Distributed Computing*, Vol. 19, No. 3, November 1993, pp. 262–270.

One of the novel features of the hypersphere mapper technique is that it can accommodate the problem of mapping more that N software tasks onto a hypercube having N processing elements (PEs). This is in contrast to other approaches found in the literature that typically assume N or fewer tasks are to be mapped onto an N PE hypercube. Another novel feature of hypersphere mapper is that it does not necessarily produce a one-to-one mapping, even for the case where there are N or fewer tasks to be mapped onto N PEs. This feature

(of producing mappings that are not necessarily one-to-one) is consistent with experience on actual systems, i.e., utilizing all available PEs does not necessarily decrease overall execution time. For example, if the volume of communication between two tasks is sufficiently high, it may be more effective to place (i.e., map) these two tasks onto the same PE in order to avoid delays associated with accessing the interconnection network for intertask communication.

After gaining an understanding and appreciation for the theoretical issues and difficulties of the hypercube embedding problem (both from the literature survey and the development of the hypersphere mapper technique) our next task involved the practical application of this understanding to a real system. It was decided to evaluate the merit of various hypercube embedding heuristics (including our proposed hypersphere mapper technique) based on performance from an actual MIMD hypercube (the nCUBE 2). Surprisingly, none of the papers in the literature that we found make comparisons among various techniques based on measured execution times from actual hypercube-based machines. Instead, competing mapping heuristics are typically compared based on the time complexity of the mapping heuristic themselves and/or the value of a simple mapping metric. An example of a mapping metric that is often used as a basis for evaluating mapping heuristics is the "average distance between pairs of communicating tasks," where the distance between a pair of mapped tasks is defined as the number of communication links between them. Although such a metric is convenient to define and use as a theoretical basis for optimization, from a practical point of view, actual performance (e.g., execution time) does not generally correlate perfectly with such a simple measure of mapping quality.

1.2 OMARS

OMARS (Optimal Mapping Alternate Routing System) is a prototype tool to aid in the evaluation of various mapping techniques on an actual system. The current version of OMARS is for operation with the nCUBE 2 hypercube-based architecture. The addition of other architectures (and supporting mapping algorithms) could, conceptually, be made in the future.

The motivation for OMARS and an overview of its functionality are presented in the present subsection; a detailed description of OMARS is given in Section 3. Also, a technical paper (co-authored with Richard C. Metzger, Loretta S. Auvil, and Chester A. Wright of Rome Laboratory and Yan Alexander Li, Olivia K. Wu, and Eduardo Asbun of Purdue University) was published entitled "OMARS: Optimal Mapping Alternate Routing System," in the *Proceedings of the Parallel Systems Fair of the 8th International Parallel Processing Symposium*, April 1994 (to appear).

One basic question that must be addressed in order to justify (if not motivate) a tool such as OMARS is "Why change the mapping?" There are at least two answers to this

question. First, it is known that the way in which software tasks are mapped onto parallel architectures can have a significant impact on delivered performance. This interaction between software and hardware, while nonexistent for sequential platforms, is one of the most basic problems to be addressed for effective execution on parallel computing platforms. Second, real software systems evolve over time. Thus, the issue of how to best map new and/or evolving multiple tasks must be addressed periodically. So, while the issue of having to address the mapping problem is nonexistent for sequential platforms, the need to do so not only once, but periodically, for parallel platforms introduces new software management problems for the case of parallel platforms.

Having acknowledged the importance of the mapping problem, one may be inclined to ask "Why not completely automate the mapping process (i.e., Why not hide it from the software engineer)?" While complete automation of the mapping process is, of course, an ideal goal; existing automatic mapping techniques fall short of this ideal. In particular, existing techniques generally attempt to optimize a simplified measure of mapping quality. Such simplified metrics for mapping quality typically ignore many important salient features associated with the assumed parallel platform. The problem is further exasperated by the reality that current parallel processing technology is not stabilizing. Thus, by the time effective automated mapping techniques could be developed for an existing parallel system technology, that technology may no longer be in use! Therefore, until parallel processing technology begins to stabilize, it is unlikely that completely automated techniques will deliver the best possible performance. We believe that tools such as OMARS can provide a "link" between users and parallel system developers that will expedite the convergence to stable parallel processing system technologies (and/or standards), thus enabling effective automated mapping techniques to be developed sooner.

The basic mapping problem is depicted in Figure 1. From the user's point of view, the mapping problem is a question of how to modify the task-to-PE map file in order to improve performance. To make use of OMARS as an aid for deciding how to best modify the task-to-PE mapping, the user must provide OMARS with the intertask communication pattern among the tasks. To assist the user in obtaining the intertask communication pattern, a tracing facility is provided for the user. Figure 2 shows that the user's multiple software tasks can be automatically instrumented to produce a trace file upon termination of execution of the multiple tasks on the parallel architecture. The trace file contains the intertask communication pattern. On-line documentation and support for the tracing facility are included with OMARS.

Once the trace file is obtained, the user can run OMARS. Figure 3 highlights the various interactions and options of OMARS. Note that the trace file serves as input to OMARS

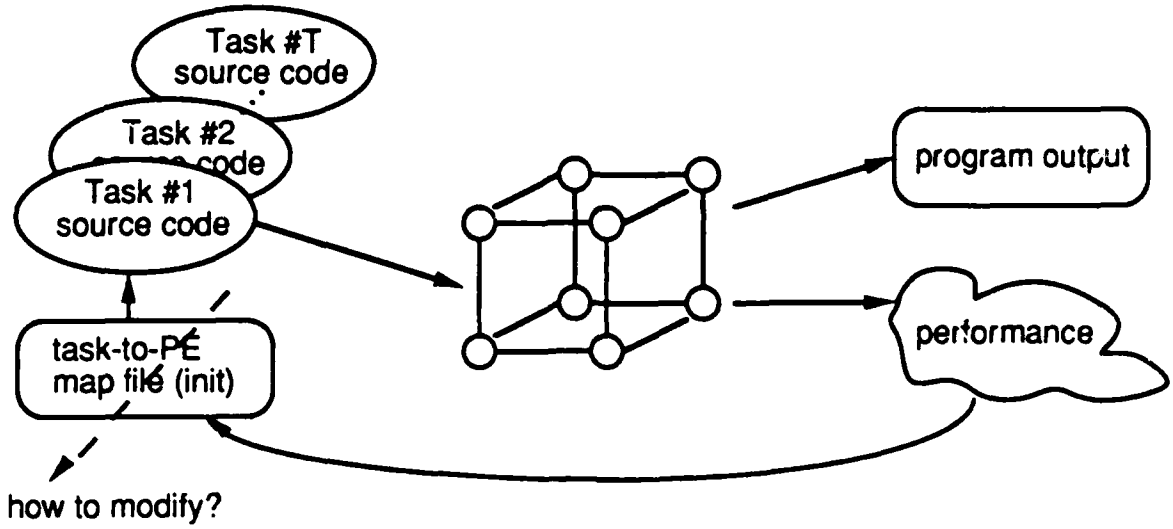

Figure 1: Illustration of the mapping problem. From the user's point of view, the mapping problem is a question of how to modify the task-to-PE map file to improve performance.

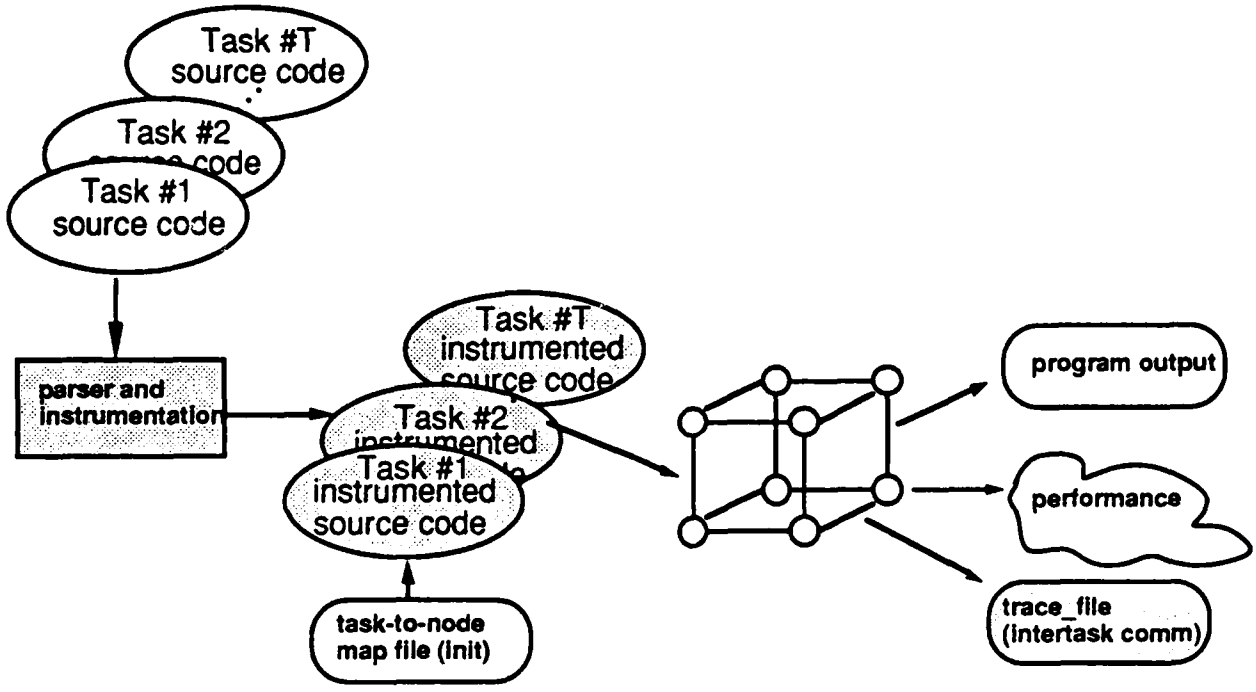

Figure 2: Tracing instrumentation is provided to capture the intertask communication pattern of the user's application.

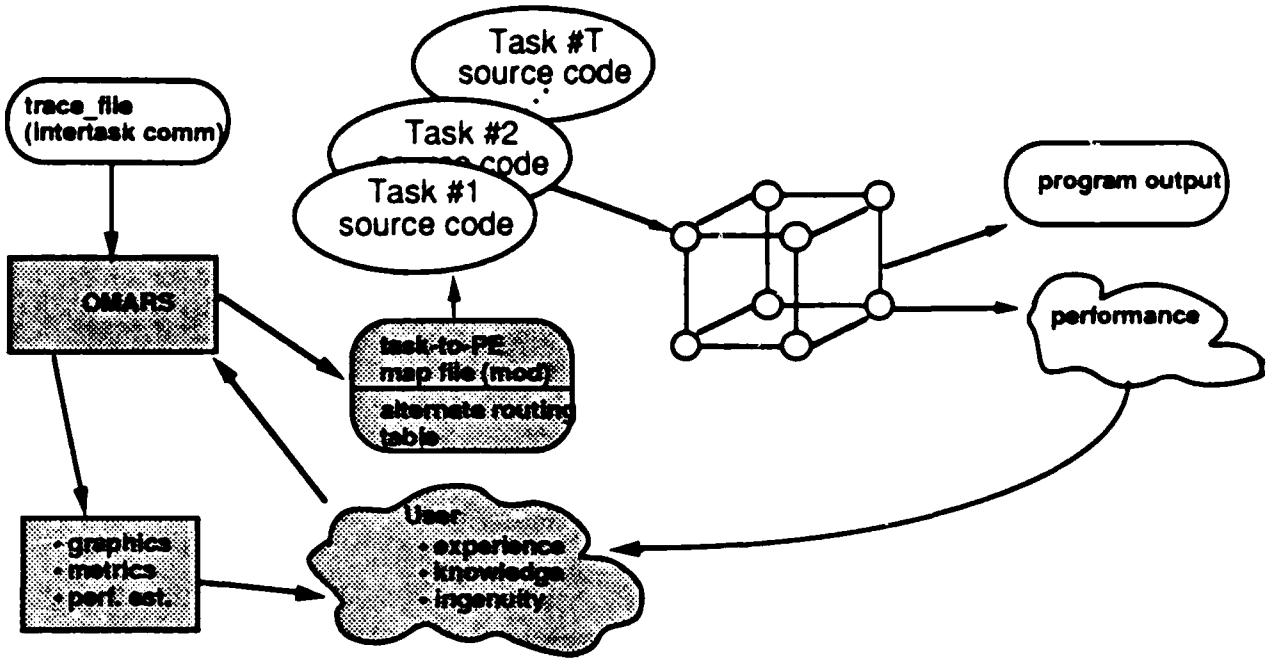

Figure 3: Overview of the interactions and options for OMARS.

and that the user provides interactive input to OMARS. The user has two sources of input: one is the output displays of OMARS and the other is the actual measured performance from the parallel architecture. Typically, the user will utilize the output from OMARS to screen-out unlikely mapping options based on the graphics, metrics, and performance estimations provided by OMARS. However, because these estimates provided by OMARS do not always perfectly predict actual performance from the parallel machine, the user can also execute the parallel tasks using selected task-to-PE map files (produced by OMARS) as a means of correlating OMARS performance estimates/metrics with actual performance. As stated earlier, this feedback to the user will enhance the user's understanding of what metrics provided by OMARS are most important for the user's particular application tasks.

1.3 The Routing Problem

Once a set of software tasks are mapped onto a parallel system, it is possible to enhance performance by altering the routes taken through the interconnection network of the parallel machine (assuming multiple routes exist). The alternate routing portion of OMARS has not been fully implemented for the nCUBE 2. The idea of implementing alternate routes is to give the user another parameter to vary in order to improve performance. The current version of OMARS supports two static routing algorithms: default and modified multiflo. The default routing scheme is the one implemented by the nCUBE 2. The modified multiflo

selects alternate routes through the hypercube (based on the given intertask communication pattern) in order to minimize contention for common network resources. In order to implement alternate routes, the operating system (OS) of the nCUBE must be modified. A license agreement has been secured with nCUBE Corporation and we are currently modifying the OS to allow for the implementation of alternate routes. With alternate routing in place, the user will be able to experiment with various mapping/routing combinations. This is interesting because the implemented mapping techniques and the multiflo routing technique aim to optimize objective functions that can be competing. Currently, the user can see the effect that alternate routing has on the performance estimates provided by OMARS; but because alternate routing is not yet implemented on the nCUBE 2, the actual (measured) effect on performance cannot be determined.

2 Hypersphere Mapper: A Nonlinear Programming Approach to the Hypercube Embedding Problem

2.1 Introduction

2.1.1 Motivation

The hypercube structure has been a popular choice for interconnecting large numbers of processing elements in parallel processing systems. Examples of commercially available parallel machines that utilize a hypercube interconnection topology include nCUBE's nCUBE 2, Connection Machine's CM2, and Intel's iPSC2.

An n-dimensional hypercube has two connected PEs along each of n dimensions for a total of $N = 2^n$ PEs. Some of the attractive features of the hypercube are: a relatively low number of incident links at each PE (PE degree $= n = \log_2 N$), a small hop distance between PEs (network diameter $= n = \log_2 N$), and a large number of alternate paths between PE pairs.

The problem addressed here is how to map a given collection of tasks onto the PEs (i.e., processing elements) of the hypercube topology so that the available communication resources are effectively utilized. The specific objective of interest is to map the tasks so that the average Hamming distance (i.e., path length), between those pairs of tasks that require communication, is minimized. This objective is certainly desirable for hypercube structures where routing is handled in a store-and-forward message passing fashion because the total delay in sending a message from source to destination is proportional to the number of links traversed. For the case of circuit-switched and virtual cut-through routing schemes, minimizing the average distance between communicating task pairs reduces the total number of communication links needed to establish all required connections and can therefore potentially reduce the latency caused by contention for a limited number of communication resources.

Effectively mapping a collection of tasks onto the PEs of a massively parallel computer is especially important when considering problem domains where the intertask communication pattern is inherently irregular and/or data dependent. For example, in the general area of computational fluid dynamics, certain applications require that the solution of a partial differential equation be approximated over an *irregular* grid of discrete points [12].

In other applications, where the tasks are *functionally independent*, the associated intertask communication pattern may also be irregular and possibly data dependent. For example, consider a large embedded information processing system (e.g., a data fusion sys-

tem) in which the input data comes from a collection of distributed sensors. The processing of the sensor data could be done using functional parallelism, i.e., one task (or perhaps a collection of subtasks) may be performing FFTs, while other tasks are involved in solving systems of linear equations, sorting integers, etc. In such a system, it is likely that the communication patterns between the functionally independent tasks will be irregular and also depend on the values of the input data supplied by the sensors.

2.1.2 Related Work

The hypercube embedding problem has been studied extensively in the past and variety of mapping heuristics and theoretical results have been reported in the literature. In reference [9], twelve hypercube mapping heuristics are evaluated in terms of mapping quality and running time (of the mapping algorithms themselves) for several classes of intertask communication patterns. In other papers, fundamental theoretical results have been established for the case where the communication patterns are assumed to have regular structures such as grids [5], trees [23], binary trees [22], and hypercubes [6].

One potential drawback of previous embedding heuristics is that they produce one-to-one mappings. In practical applications, the requirement may arise to map T tasks onto N PEs, where $T > N$; thus, many-to-one mappings are generally required. It appears that all of the known mapping heuristics (including the twelve evaluated in reference [9]) assume $T \leq N$. One way to overcome the one-to-one limitation of these mapping heuristics is to initially cluster the T tasks into N (or fewer) clusters and then execute the mapping heuristic based on the *intercluster* communication pattern. However, because the clustering problem is generally NP-complete, an additional heuristic would be required to do the initial clustering before actually executing the mapping heuristic. One of the advantages of the algorithm proposed here is that the case of $T > N$ is handled directly because the produced mapping solution is not limited to the class of one-to-one functions.

2.1.3 Overview of the Section

In Subsection 2.2, the general hypercube embedding problem is formulated in mathematical terms. A brief overview of the proposed algorithm, called hypersphere mapper, is given in Subsection 2.3. Because hypersphere mapper utilizes an iterative gradient descent technique known as the gradient projection method, a general overview of the gradient projection method is provided in Subsection 2.4. Subsection 2.5 contains the detailed description of hypersphere mapper. In Subsection 2.6, extensive simulation studies for hypersphere mapper are conducted for the case of randomly generated task communication patterns.

2.2 The Hypercube Embedding Problem

Mapping a set of T tasks, denoted $\mathcal{T} = \{0, 1, \ldots, T-1\}$, onto an $N = 2^n$ PE (i.e., n-dimensional) hypercube is viewed as a problem of determining a collection of T binary n-vectors, $z_i \in \{z : z \in \{0,1\}^n\}$, $i \in \mathcal{T}$, where the components of z_i comprise the binary address of the PE that task i is to be mapped to. For instance, for the case $n = 4$, $z_0 = [\ 1\ \ 1\ \ 0\ \ 0\ ]'$ indicates that task 0 is to be mapped onto PE 3.

Let $d_H(z_i, z_j)$ denote the Hamming distance between PEs with addresses z_i and z_j, where the Hamming distance equals the number of differing binary components associated with the vectors z_i and z_j. For example, $d_H([\ 1\ \ 0\ \ 1\ \ 0\ ]', [\ 1\ \ 1\ \ 0\ \ 0\ ]') = 2$. Let $\mathcal{W}$ denote the set of pairs of tasks that require communication. Thus, $\mathcal{W} = \{(i,j) : i,j \in \mathcal{T}\ \&$ task i communicates with task $j\}$.

Therefore, for a given set $\mathcal{W}$ (i.e., a given communication pattern), the hypercube embedding problem involves finding T binary n-vectors, denoted by $z_0, z_1, \ldots, z_{T-1}$, that minimize

$$f_H(z) = \frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} d_H(z_i, z_j). \tag{1}$$

The standard hypercube embedding formulation assumes that $T \leq N$ and that the mapping is a one-to-one function, i.e., $z_i \neq z_j$, for all $i,j \in \mathcal{T}$, with $i \neq j$. It has been proven in reference [10] that the standard embedding problem is NP-hard.

2.3 Overview of Hypersphere Mapper

The main premise of the hypersphere mapper approach is to approximate the discrete space of the hypercube, i.e., $\{z : z \in \{0,1\}^n\}$, with a continuous n-dimensional (unit radius) hypersphere, i.e., $\{x : x \in \Re^n\ \&\ ||x||^2 = 1\}$. The mapping problem in the continuous domain is to determine a collection of T real n-vectors, $x_i \in \{x : x \in \Re^n\ \&\ ||x||^2 = 1\}$, $i \in \mathcal{T}$. The advantage of the continuous domain is that techniques from nonlinear programming can be employed to solve a constrained optimization problem. In particular, by defining a continuously differentiable objective function, the values of x_i are updated in an iterative fashion by using the gradient projection technique. Geometrically, the vectors x_i are points that reside on the surface of a hypersphere in the continuous domain of $\Re^n$. The application of the iterative gradient projection technique acts to migrate these points around the surface of the hypersphere so as to minimize a desired objective function.

The objective function used by hypersphere mapper comprises two components: the first component is the average squared Euclidean distance between communicating pairs of tasks, the second component is the average of the inverted squared Euclidean distance between every pair of tasks. The first component acts to bring pairs of communicating tasks close

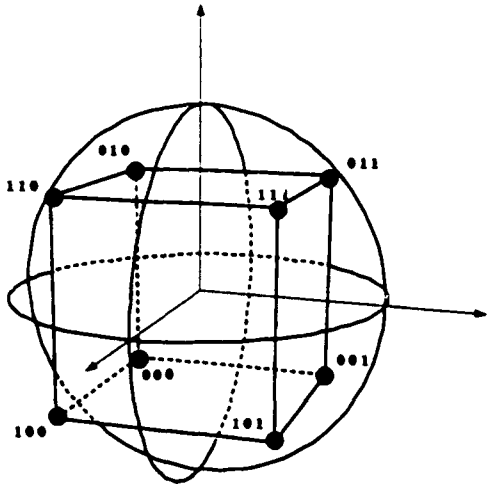

Figure 4: A geometric view of how points from the surface of a continuous hypersphere are discretized onto the PEs of a discrete hypercube. All continuous points belonging to the same sector are discretized onto the corresponding PE. For the three dimensional case shown, note that there are eight sectors and eight PEs. In the general n-dimensional case, there are 2^n sectors and and 2^n PEs.

together; the second component acts to "spread-out" the task locations (and prevent any pair of tasks from residing at the same point on the continuous hypersphere).

Once a solution is obtained on the continuous hypersphere, the solution vectors, denoted by x_i^*, are discretized onto the n-dimensional binary hypercube according to the sign of each component. For example, in three dimensions, the continuous vector $x_0^* = [\ 0.4000 \quad -0.5000 \quad 0.7681\ ]'$ discretizes to $z_0^* = [\ 1 \quad 0 \quad 1\ ]'$, which indicates that task 0 is to be mapped onto PE 5. Figure 4 depicts a geometric interpretation of how points on the surface of the continuous hypersphere are discretized onto the underlying hypercube.

2.4 The Gradient Projection Method

The gradient projection method is a iterative gradient descent technique for solving constrained optimization problems. Each iteration of the gradient projection method comprises two parts: (1) updating the values of the variables by moving in the direction of the negative gradient of the objective function and (2) orthogonally projecting the values of the variables onto the constraint space.

Consider the following general constrained optimization problem:

$$\text{minimize} \quad f(x) \tag{2}$$

$$\text{subject to} \quad g(x) = 0. \tag{3}$$

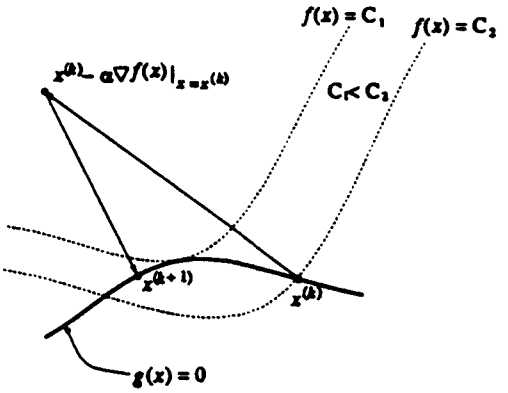

Figure 5: A geometric view of an iteration of the gradient projection method being applied to a general constrained optimization problem. The dashed lines represent constant value contours for the objective function $f(x)$, i.e., $f(x) = C_1$ and $f(x) = C_2$. The value of $x^{(k)}$ is initially updated by moving in the direction of the negative gradient, i.e., $x^{(k)}$ is updated to the point $x^{(k)} - \alpha \nabla f(x)|_{x=x^{(k)}}$. Then, the updated value is projected back to the constraint surface $g(x) = 0$.

The first part of the gradient projection method is to update x by moving in the direction of the negative gradient of f. Thus, letting $x^{(k)}$ denote the value of the vector x at iteration k, the first part of the gradient projection method is given by

$$x^{(k+1)} \leftarrow x^{(k)} - \alpha \nabla f(x)|_{x=x^{(k)}}, \tag{4}$$

where α is a positive scalar called the stepsize.

The second part of the gradient projection method is to orthogonally project $x^{(k+1)}$ onto the constraint space $g(x) = 0$. This projection operation is denoted by

$$x^{(k+1)} \leftarrow [x^{(k+1)}]^{g(x)=0}. \tag{5}$$

The mathematical description of precisely how to do the projection for general constraint surfaces shall not be included here. The interested reader is referred to [3]. A geometric view of an iteration from the gradient projection method is depicted in Figure 5.

2.5 Detailed Description of Hypersphere Mapper

The detailed description of hypersphere mapper is divided into three main parts. First, a constrained optimization problem is formulated for the problem of mapping tasks onto the surface of a continuous n-dimensional hypersphere. Second, the gradient projection technique is applied as a means of solving the proposed constrained optimization problem. Finally, the

method of discretizing the continuous task "locations" (from the solution of the constrained optimization problem) is described. After presenting the detailed description of hypersphere mapper, an illustrative example application is given. The final subsection describes a simple technique for incrementally spreading-out the hypersphere mapper solution so as to obtain more uniform mappings.

2.5.1 Formulating the Constrained Optimization Problem

In the domain of the continuous hypersphere, the underlying objective is to minimize the average squared Euclidean distance between pairs of communicating tasks, where the tasks are located on the surface of an n-dimensional hypersphere in $\Re^n$. Thus, given $\mathcal{W}$ (i.e., the set of pairs of tasks that require communication), the problem is to find T real n-vectors, $x_i \in \{x : x \in \Re^n \ \& \ ||x||^2 = 1\}$, $i \in \mathcal{T}$, that minimize

$$\frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} ||x_i - x_j||^2, \tag{6}$$

where $x_i = (\ x_{i,0} \ \ x_{i,1} \ \ \cdots \ \ x_{i,n-1})'$ and $||x_i - x_j||^2 = \sum_{\ell=0}^{n-1}(x_{i,\ell} - x_{j,\ell})^2$.

It is also desirable to enforce that $x_i \neq x_j$, for all $i, j \in \mathcal{T}$, with $i \neq j$. To enforce this constraint, the following penalty term is proposed:

$$\frac{2}{T(T-1)} \sum_{i=0}^{T-2} \sum_{j=i+1}^{T-1} \frac{1}{||x_i - x_j||^2}, \tag{7}$$

which is the average of the inverted squared Euclidean distance between all pairs of tasks. The desired objective function comes by adding Equations 6 and 7:

$$\frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} ||x_i - x_j||^2 + \frac{2}{T(T-1)} \sum_{i=0}^{T-2} \sum_{j=i+1}^{T-1} \frac{1}{||x_i - x_j||^2}. \tag{8}$$

The first term in Equation 8 acts to bring the communicating pairs of tasks close together while the second terms ensures that no two tasks reside at the same position.

Combining the objective function of Equation 8 with the constraint that the tasks must be located on the surface of an n-dimensional hypersphere results in the following constrained optimization problem:

$$\text{minimize} \ \left\{ \frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} ||x_i - x_j||^2 + \frac{2}{T(T-1)} \sum_{i=0}^{T-2} \sum_{j=i+1}^{T-1} \frac{1}{||x_i - x_j||^2} \right\} \tag{9}$$

$$\text{subject to} \ \ ||x_i||^2 = 1, \ \ \text{for all } i \in \mathcal{T}. \tag{10}$$

2.5.2 Application of the Gradient Projection Method

Note that the constrained optimization problem of Equations 9 and 10 can be expressed as:

$$\text{minimize } f(x) \tag{11}$$

$$\text{subject to } g(x) = 0, \tag{12}$$

where

$$f(x) = \left\{ \frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} ||x_i - x_j||^2 + \frac{2}{T(T-1)} \sum_{i=0}^{T-2} \sum_{j=i+1}^{T-1} \frac{1}{||x_i - x_j||^2} \right\}, \tag{13}$$

$$g(x) = \begin{pmatrix} ||x_0||^2 - 1 \\ ||x_1||^2 - 1 \\ \vdots \\ ||x_{T-1}||^2 - 1 \end{pmatrix}, \tag{14}$$

and

$$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{T-1} \end{pmatrix}. \tag{15}$$

Derivation of the gradient

Recall from Equation 4 that the first part of each iteration of the gradient projection method requires the gradient of $f(x)$, i.e., $\nabla f(x)$. By differentiating terms from $f(x)$, it can be verified that

$$\frac{\partial}{\partial x_i} \left\{ ||x_i - x_j||^2 \right\} = 2(x_i - x_j),$$

$$\frac{\partial}{\partial x_j} \left\{ ||x_i - x_j||^2 \right\} = 2(x_j - x_i),$$

$$\frac{\partial}{\partial x_i} \left\{ \frac{1}{||x_i - x_j||^2} \right\} = \frac{2}{||x_i - x_j||^4}(x_j - x_i),$$

and

$$\frac{\partial}{\partial x_j} \left\{ \frac{1}{||x_i - x_j||^2} \right\} = \frac{2}{||x_i - x_j||^4}(x_i - x_j).$$

Define $\mathcal{W}(i) = \{j : j \in \mathcal{T} \text{ \& task } i \text{ and task } j \text{ communicate }\}$. Also, define $\mathcal{T}(i) = \mathcal{T} - \{i\}$. With these definitions, the gradient of $f(x)$ is expressed as

$$\nabla f(x) = \begin{pmatrix} \frac{2}{|\mathcal{W}|} \sum_{j\in\mathcal{W}(0)} (x_0 - x_j) \\ \frac{2}{|\mathcal{W}|} \sum_{j\in\mathcal{W}(1)} (x_1 - x_j) \\ \vdots \\ \frac{2}{|\mathcal{W}|} \sum_{j\in\mathcal{W}(T-1)} (x_{T-1} - x_j) \end{pmatrix} + \begin{pmatrix} \frac{4}{T(T-1)} \sum_{j\in\mathcal{T}(0)} \frac{(x_j - x_0)}{||x_0 - x_j||^4} \\ \frac{4}{T(T-1)} \sum_{j\in\mathcal{T}(1)} \frac{(x_j - x_1)}{||x_1 - x_j||^4} \\ \vdots \\ \frac{4}{T(T-1)} \sum_{j\in\mathcal{T}(T-1)} \frac{(x_j - x_{T-1})}{||x_{T-1} - x_j||^4} \end{pmatrix}. \tag{16}$$

Thus, the first part of the iteration for the gradient projection method (i.e., moving in the direction of the negative gradient) is given by

$$x^{(k+1)} \leftarrow x^{(k)} - \alpha \nabla f(x)|_{x=x^{(k)}}, \tag{17}$$

where $\nabla f(x)|_{x=x^{(k)}}$ is computed according to Equation 16.

Derivation of the projection

After the update of Equation 17 is computed, the second part of the gradient projection method requires that the resultant $x^{(k+1)}$ be projected back to the surface $g(x) = 0$. This amounts to simply normalizing each x_i to be unit length (in the Euclidean sense). Thus, the projection operator is defined by

$$[x]^{g(x)=0} = \begin{pmatrix} \frac{x_0}{\|x_0\|} \\ \frac{x_1}{\|x_1\|} \\ \vdots \\ \frac{x_{T-1}}{\|x_{T-1}\|} \end{pmatrix}. \tag{18}$$

So, the second part of the iteration for the gradient projection method is given by

$$x^{(k+1)} \leftarrow [x^{(k+1)}]^{g(x)=0}, \tag{19}$$

where $[x^{(k+1)}]^{g(x)=0}$ is computed according to Equation 18.

After a sufficient number of iterations, the gradient projection method converges to a fixed-point, i.e., the vector $x^{(k)}$ converges to a fixed point x^* as k increases.

Discretizing onto the hypercube

The final part of hypersphere mapper is to convert the solution vector x^*, which has continuous real components, to z^*, which has discrete components in the set $\{0, 1\}$. The conversion is done by simply discretizing each component of x (to either zero or one) according to their signs. Formally, for any $t \in \Re$, the unit step function is defined as

$$u(t) = \begin{cases} 1 & \text{if } t \geq 0 \\ 0 & \text{if } t < 0 \end{cases}$$

Therefore, the discretization is defined by

$$z^* = u(x^*),$$

where it is understood that $u(\cdot)$ is applied to each component of x^*.

2.5.3 An Illustrative Example

The goal here is to illustrate how the task locations move around the surface of the hypersphere from one iteration to the next. Of particular interest is to show how the relative

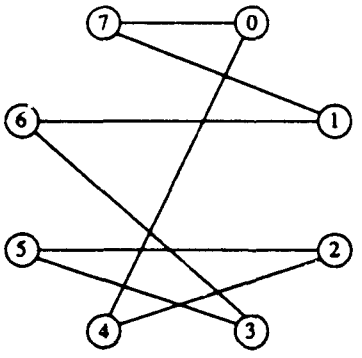

Figure 6: Graphical view of the example task communication pattern $\mathcal{W} = \{(0,4),(0,7),(1,7),(1,6),(2,4),(2,5),(3,5),(3,6)\}$.

locations of the task (for a given communication pattern, $\mathcal{W}$) affect the iterative migration to the fixed point.

Consider the problem of mapping eight tasks, $\mathcal{T} = \{0,1,\ldots,7\}$, onto the PEs of a 3-dimensional hypercube. Assume the communication pattern is $\mathcal{W} = \{(0,4),(0,7),(1,7),(1,6),(2,4),(2,5),(3,5),(3,6)\}$. A graphical view of the assumed task communication pattern, i.e., $\mathcal{W}$, is shown in Figure 6. The resulting iteration data from hypersphere mapper is given in Table I.

From Table I, note that the initial values of the continuous location vectors (i.e., the values of $x_i^{(0)}$, $i = 0,1,\ldots,7$) were selected so that task i is (initially) mapped to PE i, $i = 0,1,\ldots,7$. After each iteration k, the resulting values of the continuous location vectors, $x_i^{(k)}$, and the associated discretized location vectors, $z_i^{(k)}$, are given. Also, the corresponding values of the continuous objective function, $f(x^{(k)})$ (refer to Equation 13), and the associated average Hamming distance, $f_H(z^{(k)})$ (refer to Equation 1), are given. The fact that the value of $f_H(x^{(k)})$ decreases as k increases indicates that the continuous objective function does indeed capture the essence of the discrete mapping problem.

Note that the task to PE mapping produced by hypersphere mapper (taken from iteration $k = 10$ of Table I) is:

task 0 $\rightarrow$ PE 4
task 1 $\rightarrow$ PE 7
task 2 $\rightarrow$ PE 0
task 3 $\rightarrow$ PE 3
task 4 $\rightarrow$ PE 4
task 5 $\rightarrow$ PE 1
task 6 $\rightarrow$ PE 7
task 7 $\rightarrow$ PE 5,

which is *not* a one-to-one mapping because two tasks are mapped to PE 4 and two tasks are mapped to PE 7 (and no tasks are mapped to PE 2 nor PE 6). The average Hamming distance between communicating tasks for this mapping is 0.75.

Table 1: Iterations from hypersphere mapper.

k	x_0^k z_0^k	x_1^k z_1^k	x_2^k z_2^k	x_3^k z_3^k	x_4^k z_4^k	x_5^k z_5^k	x_6^k z_6^k	x_7^k z_7^k	$f(x^k)$ $f_H(z^k)$
0	−0.66 −0.56 −0.50 0 0 0	0.61 −0.75 −0.26 1 0 0	−0.34 0.07 −0.94 0 1 0	0.66 0.55 −0.50 1 1 0	−0.03 −0.80 0.59 0 0 1	0.09 −0.91 0.41 1 0 1	−0.45 0.61 0.65 0 1 1	0.61 0.55 0.57 1 1 1	4.12 2.25
1	−0.66 −0.68 −0.32 0 0 0	0.78 −0.61 −0.14 1 0 0	−0.36 −0.16 −0.92 0 0 0	0.69 0.59 −0.40 1 1 0	−0.51 0.15 0.84 0 1 1	0.46 −0.62 −0.63 1 0 0	−0.33 0.70 0.63 0 1 1	0.73 0.41 0.54 1 1 1	3.06 2.00
2	−0.73 −0.68 −0.05 0 0 0	0.68 −0.42 0.22 1 0 1	−0.38 −0.17 −0.91 0 0 0	0.68 0.61 −0.42 1 1 0	−0.71 −0.15 0.69 0 0 1	0.35 −0.47 −0.81 1 0 0	−0.07 0.87 0.48 0 1 1	0.81 0.21 0.54 1 1 1	2.31 1.50
3	−0.72 −0.69 0.13 0 0 1	0.89 −0.32 0.31 1 0 1	−0.46 −0.19 −0.87 0 0 0	0.64 0.63 −0.45 1 1 0	−0.82 −0.24 0.53 0 0 1	0.38 −0.36 −0.85 1 0 0	0.14 0.89 0.44 1 1 1	0.78 0.11 0.61 1 1 1	2.06 1.00
9	−0.39 −0.70 0.60 0 0 1	0.91 −0.01 0.42 1 0 1	−0.62 −0.27 −0.74 0 0 0	0.57 0.59 −0.58 1 1 0	−0.94 −0.33 0.07 0 0 1	0.19 −0.05 −0.98 1 0 0	0.66 0.72 0.20 1 1 1	0.33 −0.34 0.88 1 0 1	1.38 0.75
10	−0.39 −0.68 0.61 0 0 1	0.89 0.01 0.45 1 1 1	−0.62 −0.27 −0.74 0 0 0	0.56 0.58 −0.59 1 1 0	−0.93 −0.33 0.05 0 0 1	0.16 −0.03 −0.99 1 0 0	0.68 0.71 0.18 1 1 1	0.32 −0.36 0.88 1 0 1	1.37 0.75

It can be verified (through an exhaustive search, for example) that an optimal *one-to-one* mapping is given by:

task 0 → PE 5
task 1 → PE 6
task 2 → PE 0
task 3 → PE 3
task 4 → PE 4
task 5 → PE 1
task 6 → PE 2
task 7 → PE 7.

The above optimal one-to-one mapping has an associated average Hamming distance of 1.00. Thus, for this example, the solution from hypersphere mapper produces a mapping that is superior, in terms of the average Hamming distance, to the best possible one-to-one mapping (i.e., 0.75 versus 1.00) at the "expense" of not being one-to-one.

2.5.4 Converting Hypersphere Mapper Solutions into Uniform Mappings

An intuitive approach is introduced here for modifying the continuous solution produced by hypersphere mapper so that after discretization, the resulting mapping will be uniform (or to within a specified degree of closeness to being uniform). The basic idea of the approach is the incrementally spread continuous task locations out of "over-populated" sectors and into nearby "under-populated" sectors. The detailed description of the approach requires the following definition.

In the space of a continuous n-dimensional hypersphere, let $\underline{c_i} = (\; c_{i,0} \;\; c_{i,1} \;\; \cdots \;\; c_{i,n-1} \;)' \in \{x : x \in \Re^n \;\&\; ||x||^2 = 1\}$ denote the center of sector i, which is defined for all $i = 0, 1, \ldots, 2^n - 1$ as follows:

$$c_{i,j} = \begin{cases} \frac{1}{\sqrt{n}}, & \text{if bit } j \text{ of } i \text{ is unity} \\ -\frac{1}{\sqrt{n}}, & \text{if bit } j \; i \text{ is zero.} \end{cases}$$

For instance, for the 3-dimensional case, $c_0 = (\; -\frac{1}{\sqrt{3}} \;\; -\frac{1}{\sqrt{3}} \;\; -\frac{1}{\sqrt{3}} \;)'$ and $c_6 = (\; -\frac{1}{\sqrt{3}} \;\; \frac{1}{\sqrt{3}} \;\; \frac{1}{\sqrt{3}} \;)'$. The center vectors define the geometric centers for each of the 2^n sectors associated with an n-dimensional hypersphere.

The algorithm for incrementally spreading out the continuous solution from hypersphere mapper operates as follows. First, those sectors having strictly more than $\lceil \frac{T}{N} \rceil$ tasks residing in them are flagged as being "over-populated" and those having no more than $\lfloor \frac{T}{N} \rfloor$ tasks residing in them are flagged as being "under-populated." The spreading procedure takes place in n consecutive phases, denoted by phase 1 through phase n. During phase i, those position vectors residing in over-populated sectors that are within a Euclidean distance of $2\sqrt{\frac{i}{n}}$ from the center of an under-populated sector are moved to that under-populated sector. Immediately after a position vector is moved from an over-populated sector to an

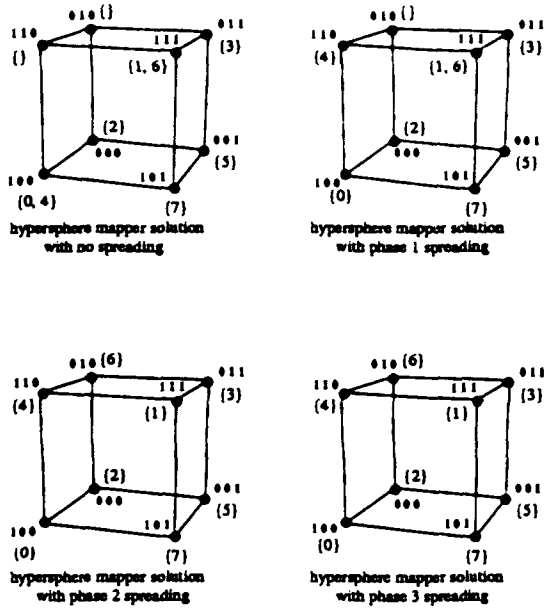

Figure 7: Applying the spreading procedure to convert the solution from hypersphere mapper into a uniform (in this case, one-to-one) function. The notation "$\{A, B\}$" beside PE address "*abc*" means that tasks A and B are mapped onto the PE labeled *abc*.

under-populated sector, the status of the associated sectors' population is updated. Because the diameter of the continuous hypersphere is 2 (in the Euclidean sense), after phase n of the procedure, it is clear that there will be no over-populated nor under-populated sectors. Also, the mapping associated with phase j of the spreading procedure is generically a more uniform mapping than the one associated with with phase i, for all $j > i$.

The spreading procedure was applied to the illustrative example of the previous subsection (i.e., the problem of mapping 8 tasks onto 8 PEs assuming the communication pattern $\mathcal{W}$ of Figure 6). The results of applying the spreading procedure are depicted in Figure 7. Without application of the spreading procedure, note that tasks 0 and 4 are both mapped to PE 4, tasks 1 and 6 are both mapped to PE 7, and no tasks are mapped to either PE 2 nor 6. Phase 1 of spreading moves task 4 from the over-populated PE 4 onto the under-populated PE 6. Phase 2 of spreading moves task 6 from the over-populated PE 7 onto the under-populated PE 2.

2.6 Evaluation of Hypersphere Mapper

2.6.1 Comparison with Other Hypercube Embedding Heuristics

In reference [GSG89], evaluations and comparisons (based on simulation studies) are reported for twelve hypercube embedding heuristics. Several classes of task communication patterns

are assumed, including random patterns, permuted hypercube patterns, and tree patterns. Of the heuristics evaluated (and for all of the assumed task communication patterns), the simulated annealing heuristic developed in [9] produced mappings with the smallest average Hamming distance.[1] In contrast, a greedy algorithm of [8] generally produced the poorest mappings (excluding the so-called default or random mapping schemes, which map tasks to PEs independent of the given task communication pattern). It is also reported that the (typical) running time for the simulated annealing algorithm is around four orders of magnitude larger than that of the greedy algorithm. Both the quality of the mappings and the running times associated with the other ten heuristics were shown to lie within the bounds defined by the simulated annealing and greedy algorithms. For a detailed description, evaluation, and comparison of the twelve heuristics, refer to [9].

In the present study, simulation studies are used to provide an indication of how the quality of mappings produced by hypersphere mapper compare to the quality of mappings produced by the other known embedding heuristics. The simulation study carried out in this subsection involves mapping 128 tasks onto the 128 PEs of a 7-dimensional hypercube. The assumed communication pattern is generated by selecting task source-destination pairs at random. The expected number of randomly selected source-destination pairs is set at 448, i.e., $E[|\mathcal{W}|] = 448$. This exact same simulation study was one of the experiments used in evaluating the heuristics in [9].

The results of simulation studies for the hypersphere mapper, simulated annealing, greedy, and default algorithms (averaged over 100 runs) are summarized in Table II. The column labeled $\hat{f}_H(z)$ denotes the average Hamming distance between mapped communicating tasks. The column labeled $\hat{\sigma}^2_{\text{pmn}}$ denotes the sample variance of the number of processes mapped to each node, which is defined by

$$\hat{\sigma}^2_{\text{pmn}} = \frac{1}{N} \sum_{i=0}^{N} \left(\text{pmn}(i) - \frac{T}{N} \right)^2 ,$$

where $\text{pmn}(i)$ denotes the number of tasks mapped to PE i. The term $\frac{T}{N}$ represents the average number of PEs mapped to each PE, which equals unity for the simulation of this subsection because there are $T = 128$ tasks being mapped onto $N = 128$ PEs. (In general, the average number of tasks mapped onto each PE equals $\frac{T}{N}$, regardless of the mapping).

From Table II, note that the value of $\hat{\sigma}^2_{\text{pmn}}$ is nonzero only for the hypersphere mapper because the mappings produced by hypersphere mapper are not (generically) one-to-one. On the other hand, the competing algorithms always produce one-to-one mappings and thus have zero values for $\hat{\sigma}^2_{\text{pmn}}$ (i.e., exactly one task is mapped to each PE, thus the variance of

[1] Refer to [18] for general background material on the simulated annealing approach.

Table 2: Comaprison between hypersphere mapper and other heuristics: 128 tasks onto 128 PEs with $E[|\mathcal{W}|] = 448$.

Algorithm	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\text{pmn}}$
Hypersphere Mapper	1.889	1.68
Simulated Annealing	2.042	0.00
Greedy	2.867	0.00
Default	3.503	0.00

the number of tasks mapped to each PE is zero). The results tabulated for the simulated annealing and greedy algorithms were taken from reference [9].[2] The last row of Table II gives the results of the default mapping algorithm, which simply maps task i to PE i, for all $i = 0, 1, \ldots, 127$.

With regard to average Hamming distance, the mappings produced by hypersphere mapper are indeed superior to those produced by the simulated annealing algorithm (i.e., 1.889 versus 2.042). However, the mappings produced by hypersphere mapper are not generically one-to-one; thus, the corresponding value of $\hat{\sigma}^2_{\text{pmn}}$ is not equal to zero.

Table III shows the effect of applying the spreading procedure to the hypersphere mapper solution. The average Hamming distance of the mapping produced after spreading phase 1 is slightly better than that of the simulated annealing algorithm (i.e., 2.020 versus 2.042) and the value of the variance of the number of tasks mapped to each PE, relative to the hypersphere mapper solution with no spreading, is reduced by around 60% (i.e., from 1.68 down to 1.07). It is also noted that the *one-to-one* mapping produced after spreading phase 7 is significantly better than the mapping produced by the greedy algorithm (i.e., 2.587 versus 2.867).

2.6.2 The Case of Having More Tasks than PEs

A primary attribute of hypersphere mapper is that problems involving more tasks than PEs (i.e., $T > N$) can be solved directly. In contrast, the competing hypercube embedding heuristics (including the twelve evaluated in [9]) require $T \leq N$.

Simulation results are reported here for the case of mapping 256 tasks onto 64 PEs. The assumed task communication patterns are randomly generated (with a specified expected number of source-destination pairs). Simulation studies are conducted for cases where the expected number of source-destination pairs is 128 256, 512, and 1024. Table IV shows

[2]The greedy algorithm was coded and simulated using our randomly generated communication patterns. The value of average Hamming distance produced by our simulation of the greedy algorithm was within 2% of the value reported in [9].

Table 3: Application of the spreading procedure to hypersphere mapper: 128 tasks onto 128 PEs with $E[|\mathcal{W}|] = 448$.

spreading	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\text{pmn}}$
none	1.889	1.68
phase 1	2.020	1.07
phase 2	2.283	0.40
phase 3	2.440	0.15
phase 4	2.524	0.06
phase 5	2.558	0.03
phase 6	2.580	0.01
phase 7	2.587	0.00

Table 4: Hypersphere mapper results: 256 tasks onto 64 PEs.

spreading	$E[\|\mathcal{W}\|] = 128$		$E[\|\mathcal{W}\|] = 256$		$E[\|\mathcal{W}\|] = 512$		$E[\|\mathcal{W}\|] = 1024$	
	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\text{pmn}}$	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\text{pmn}}$	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\text{pmn}}$	$\hat{f}_H(z)$	$\hat{\sigma}^2_{\text{pmn}}$
none	0.619	3.06	0.852	4.10	1.340	4.16	1.763	7.73
phase 1	0.850	0.45	1.000	0.87	1.433	1.28	1.850	4.05
phase 6	0.973	0.00	1.168	0.00	1.598	0.00	2.110	0.00

the results of the simulation studies in terms of average Hamming distance, $\hat{f}_h(z)$, and the sample variance of the number of tasks mapped onto each PE, $\hat{\sigma}^2_{\text{pmn}}$. As would be expected, the average Hamming distance increases as the expected number of source-destination pairs increases. Also, for a fixed value of the expected number of source-destination pairs, the average Hamming distance increases with successive applications of the spreading procedure. Simulations for a simple default mapper were also conducted, where the default mapper simply mapped task i onto PE j, where j is defined by masking (to zero) all but lower six bits of i. The simulations results for the default mapper were $\hat{f}_H(z) \approx 3.00$ and $\hat{\sigma}^2_{\text{pmn}} = 0.00$, for all cases.

2.7 Conclusions

2.7.1 Summary

A new hypercube embedding algorithm called hypersphere mapper was introduced. The hypersphere mapper approach is novel in that it utilizes techniques from nonlinear programming as a means of solving the mapping problem. In contrast, competing mapping algorithms are based on combinatoric heuristics. An important advantage of hypersphere mapper over existing mapping heuristics is that the case of mapping more that N tasks onto

N PEs is solved directly. In contrast, the competing heuristics are limited to problems where the number of tasks is no larger than the number of PEs.

The variance of the number of tasks mapped to each PE is introduced as a means of quantifying the measure of "computational load balance" associated with a particular mapping. The average distance between mapped communicating tasks is used to measure the communication load requirement on the interconnection network. Experiences with existing hypercube multiprocessors have indicated that latencies caused by contention for the communication resources are often more of a throughput bottleneck than the uniformity of the computational load balance. Hypersphere mapper offers a means of striking a practical balance between these two competing factors.

2.7.2 Ongoing and Future Work

By altering, slightly, the proposed objective function for hypersphere mapper, the resultant mappings can be tuned to have desired characteristics. For instance, by simply multiplying the average Euclidean distance component of the objective function (refer to Equation 9) by a scalar weighting factor, say γ, the nature of the resultant mappings can be changed. In particular, the mappings produced with $\gamma > 1$ tend to have smaller values of $\hat{f}_H(z)$ and larger values of $\hat{\sigma}^2_{\text{pmn}}$ (with no spreading), than the mappings produced with $\gamma \leq 1$. Also, the objective function can easily be adopted to accommodate a weight associated with each element in $\mathcal{W}$, which corresponds to the frequency with which the associated pair of tasks request communication. Thus, placing an increased penalty on separating those task pairs that communicate frequently. Work is currently underway in evaluating a variety of different objective functions.

For cases where T^2 is large relative to $|\mathcal{W}|$, the dominant computational time associated with each of the gradient projection iterations comes in computing the component of the gradient associated with the average inverted Euclidean distance between *all pairs* of tasks (refer to Equations 9 and 16). Instead of averaging the inverted Euclidean distance over all pairs of tasks, an average could be taken over a smaller set of pairs of tasks. The set $\mathcal{W}$ appears to be a natural candidate. Another possibility is to average over a randomly generated collection of task pairs. Some preliminary studies along these lines are currently underway. Work is also currently underway in integrating hypersphere mapper into a software engineering tool for high-performance computing systems.

3 OMARS: Optimal Mapping Alternate Routing System

3.1 Introduction

3.1.1 Background

With rapid advances being made in technology for information acquisition and processing, the Department of Defense has recognized the potential benefit in shifting certain computationally intensive applications from sequential to parallel platforms [19]. However, the potential performance gains associated with using parallel machines may be offset by the generic difficulty (i.e., extensive time and effort) required to effectively port applications onto current high-performance parallel architectures. Unlike the hardware and software technologies associated with sequential platforms, which are becoming more standardized and appear to be stablizing, the status of hardware and software technology for parallel systems is rapidly changing.

When executing code on sequential computers, performance is relatively independent of where in memory the software tasks (i.e., code and/or data blocks) are initially stored.[3] However, the specific mapping used to assign multiple tasks onto the processing elements of a distributed memory parallel machine can have a significant impact on performance. A poor mapping of tasks onto the processing elements of a parallel machine may, for example, result in excessive contention for some of the communication links of the interconnection network. Such contention can cause some messages to experience substantial delays as they are transmitted through the network, which in turn may cause excessive idle times for the associated destination processors. When processors spend an excessive amount of time waiting for messages from other processors (instead of doing useful computation), then the overall performance of the system can be degraded. Therefore, while the overall performance of executing large software systems on sequential machines depends primarily on the characteristic speed of the underlying hardware, the performance of executing computationally intensive applications on parallel platforms generally depends on how well the software application and parallel hardware are "matched."

Large software systems, such as those that exist in complex military systems, are often conceptualized, designed, and coded as a collection of interdependent computational tasks. A natural architectural choice for executing a collection of tasks in parallel—where the

[3]Technically, sequential machines with cache memory do improve performance by moving active/inactive code blocks into/out of a high-speed cache memory space, but this is generally done dynamically and is not controlled directly by the programmer.

functionalities of the tasks may be mutually independent—is to execute the tasks on the processors of an MIMD (multiple instruction stream - multiple data stream) machine [11]. MIMD systems, by definition, are capable of executing multiple independent threads of control, one or more on each processor. Knowledge of the communication pattern among the tasks can be used to decide to which processor each of the tasks should be mapped. For example, for the case where there are more tasks than processors, it may be beneficial to map clusters of tasks that communicate frequently onto a common processor in order to reduce traffic on the interconnection network and minimize the effect of latency associated with passing messages through the network. An effective mapping of tasks to processors may also account for other aspects such as balancing the computational load assigned to each processor and minimizing network contention for those pairs of tasks that do communicate through the interconnection network.

Because the interconnection networks employed by different parallel machines have a wide range of characteristics and properties [21], the best mapping strategy (for a given application) onto a particular architecture may not be the best strategy for another architecture. Formal methodologies have been proposed for automating the process of mapping tasks onto the processors of various classes of parallel architectures. For example, the so-called hypercube embedding problem (which involves mapping software tasks onto the processors of architectures with a hypercube topology) has been studied extensively in the literature [9]. Mapping problems are usually formulated as combinatoric optimization problems, and often these formulations are NP-Hard, for example see [10].

A common objective used for mapping tasks onto architectures that have point-to-point network topologies, such as hypercubes and meshes, is the minimization of the average distance between pairs of communicating tasks (where the distance is defined as the minimum number of network links that must be traversed to send messages between a communicating pair of tasks). While this objective makes intuitive sense under the assumption of store-and-forward switching [7], in which the delay of a message is proportional to the number of traversed links, it may or may not be an important measure under the assumption of virtual cut-through switching [17]. For the case of virtual cut-through switching, a more fundamental concern may be to minimize link contention, which is more accurately gauged by the maximum link usage factor (defined to be the maximum number of times any single link is traversed). There is generally some degree of overlap between various objective functions. For instance, minimizing the average distance between communicating tasks also minimizes the total number of links needed for all required processor-to-processor connections, which in turn may tend to decrease the maximum link usage factor. That is, if less links are used for all required connections, then (intuitively) link contention is less likely.

Judging the merit of a mapping heuristic by only examining the associated values of the underlying objective function (e.g., average and/or worst case objective function values) does not necessarily provide accurate insight into how well the mappings produced by the heuristic will perform when implemented on a real parallel machine. This is because the objective functions that heuristics aim to minimize typically do not fully capture the complexity and interdependence of the many factors that must be considered in order to accurately measure the quality of a mapping. One of the important features of OMARS is that it allows the user to experiment with various mapping strategies in order to quickly determine a mapping that will perform well when actually implemented on a specified parallel machine.

3.1.2 Motivation for OMARS

In military systems, the software development process for sequential platforms is a well-defined and standardized process. This process, defined in MIL-STD2167A, is based on the waterfall lifecycle model and consists of several major phases including requirements analysis, specification, design, coding, testing, integration, and maintenance. Because the operating systems, languages, compilers, and architectures for sequential systems are becoming more standardized, the porting of application software from one sequential platform to another is usually not a major issue.

In contrast to sequential software development, a standardized process for developing parallel software does not yet exist. It is difficult to directly apply the sequential model for software development to parallel systems because there are certain issues in developing parallel software that do not arise when developing sequential software. One important aspect of software development for parallel systems during the integration phase, is the notion of how to map multiple tasks of parallel software onto the processors of a parallel architecture.

The theoretical and technical difficulties associated with understanding, modeling, and accounting for the many interdependent factors that influence how to best map a *given* set of computational tasks onto the processors of a *given* parallel machine are compounded by the fact that real software systems evolve over time, and thus the issue of how to best map new and/or evolved multiple tasks onto a given machine must be addressed periodically. Also, hardware technology for parallel processing systems is rapidly changing, which often makes the network characteristics of newly introduced machines substantially different from their predecessors. The rapid changes in technology for parallel processing (i.e., lack of a standard) have made it very difficult for the research community to develop systematic and provably sound techniques for automating the mapping of software tasks onto parallel architectures.

The following points represent key factors that initially motivated (and continue to in-

fluence) the development of OMARS: (1) the mapping strategy used to assign tasks to the processors of distributed memory architectures can have a significant impact on delivered performance, (2) systematic techniques for mapping tasks onto parallel architectures are often based on optimizing a simplified metric associated with the assumed architecture, and thus actual performance from the parallel machine may not correlate perfectly with optimized values from the assumed objective function, (3) until the technology for parallel processing begins to stabilize, it is unlikely that completely automated techniques for mapping will be able to provide mappings that always deliver the best possible performance, (4) the problem of mapping will introduce new organizational and/or software management burdens when large evolving software systems are targeted to parallel platforms, and (5) a need for an environment to make currently available mapping heuristics more accessible, practical, and easier to evaluate and compare.

3.1.3 Organization of the Section

This section is organized in the following manner. Subsection 3.2 includes an overview of OMARS and descriptions of its main subsystems. Subsection 3.3 goes through an example user session of OMARS. Subsection 3.4 gives a brief description of a successfully implemented and very effective distributed configuration management scheme that was used to develop the software for OMARS (software development for OMARS was divided between two geographically distributed sites). Subsection 3.5 reviews the current status of OMARS and outlines ideas for future versions.

3.2 Description of OMARS

3.2.1 Overview of OMARS

OMARS is a prototype tool designed to aid the software engineer during various phases of the development of parallel software. The tool is especially well-suited for large software applications where functional parallelism is employed to parallelize computations. The OMARS tool is designed to alleviate some of the organizational problems associated with assigning computational tasks onto the processors of a parallel machine, and to provide the user with easy to understand graphical interfaces. The OMARS tool provides a "plug-and-fit" environment for executing and evaluating sophisticated mapping algorithms for solving various formulations of the mapping problem. On platforms where the interconnection network has multiple paths between processor pairs, the tool also provides routing solutions (from an optimal routing algorithm) that could be used to implement alternate routes through the interconnection network of the assumed parallel architecture.

The input to OMARS is a weighted intertask communication graph. A tracing facility is provided for capturing the intertask communication graph for parallel programs written in either the MIMD or SPMD (single program - multiple data) modes of parallelism. For a given intertask communication graph, the user can interactively experiment with various combinations of mapping and routing strategies by selecting algorithms from the mapping and routing subsystem menus of OMARS. For each mapping/routing combination selected by the user, the tool generates a window containing a graphical view of the machine's architecture providing the user with an overall visual perception of how effectively the architecture's resources are utilized. Numerically-based metrics are also displayed within the window of each architecture graph.

OMARS currently runs on a Sun 4 workstation. The OMARS graphical displays are written with the X-windows system software to provide the greatest possible portability across today's popular workstations. It currently provides graphical displays of architectures having up to 64 processing nodes and can provide general mapping and routing support for systems having up to 1024 processing nodes.

The functionality of the OMARS tool is divided into five parts: the tracefile generator, the mapping subsystem, the routing subsystem, the graphical displays, and the taskfile generator. These subsystems are described in greater detail in the following subsections. The relationships and flow of information among these subsystems are shown in Figure 8.

3.2.2 The Tracefile Generator

In order to determine the communication pattern among software tasks, parallel code can be instrumented to capture (i.e., trace) the intertask communication during execution. Because the intertask communication pattern is independent of the task-to-processor mapping, the initial mapping used for the purpose of tracing the intertask communication pattern from the parallel architecture can be arbitrary. After execution, the intertask communication pattern is written to a file, which is called a tracefile. Given the traced intertask communication information contained in the tracefile, OMARS can provide a graphical view of the communication pattern among the tasks in a task graph. Also, this intertask communication pattern is used as input information to the mapping subsystem.

The information recorded in the tracefile includes: the number of tasks, the number of processors, the processor to which each task was initially assigned (for the purposes of the trace), and the set of tasks to which each task sends messages. The intertask communication data is gathered by monitoring all messages sent by all tasks during execution of the parallel program on the parallel machine. Once the parallel program has completed execution, the traced information (which is stored in processor memory) is written to the tracefile. This

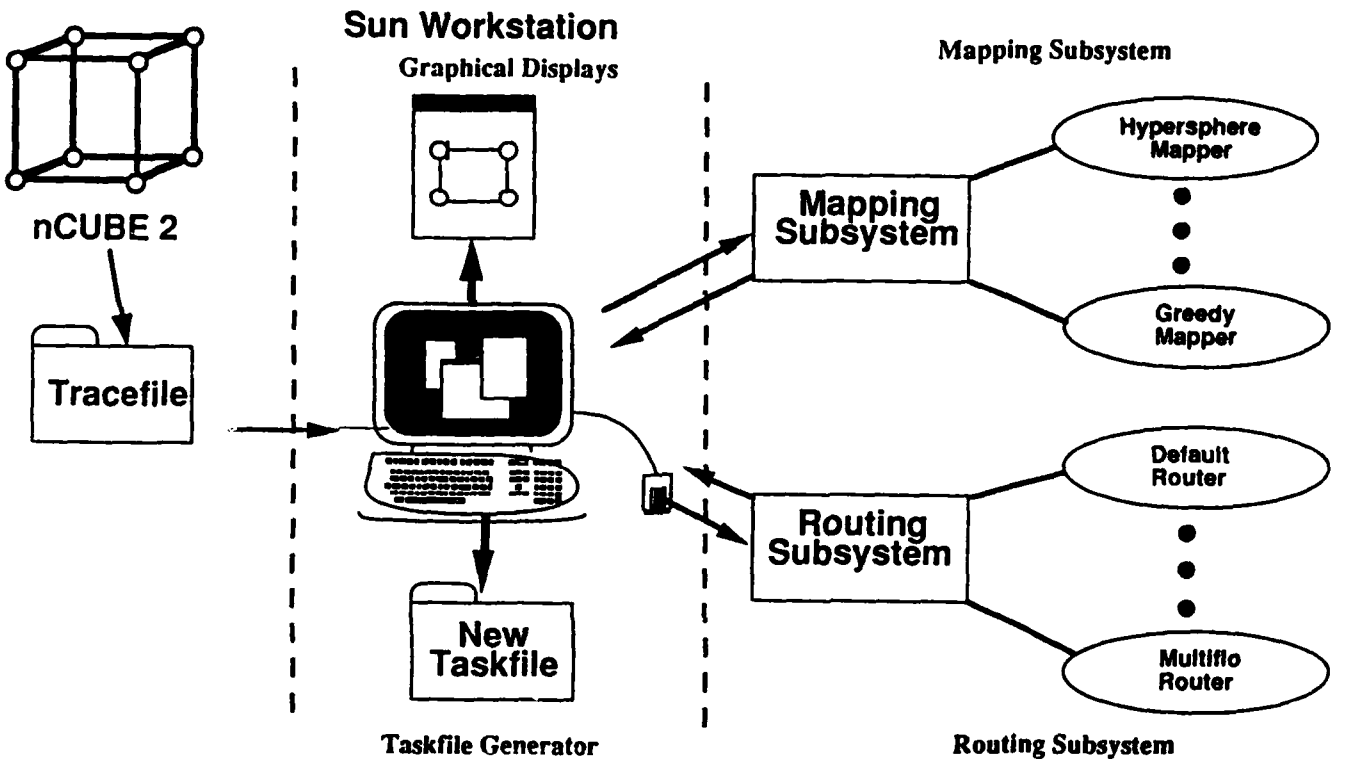

Figure 8: System overview of OMARS.

tracing procedure introduces very little intrusiveness to the execution, however, the scalability of such an approach is obviously limited by the amount of memory available to each processor. This type of memory storing approach for tracing has also been used in PICL [13, 14] producing information that can be used with the ParaGraph visualization tool [15].

The current version of the tracing facility was developed to support the nCUBE 2 parallel processing system [20]. Both the MIMD and SPMD modes of programming for the nCUBE 2 are supported.

In the MIMD mode of programming on the nCUBE, each of say T tasks are coded as T independent programs. These T programs are compiled and mapped onto the processors of the nCUBE using a user specified mapfile. The mapfile defines to which processor each program (i.e., task) is to be mapped. Sending a message from one task to another is accomplished by executing the communication construct `nwrite()` at the sending task and `nread()` at the receiving task. The arguments of the `nwrite()` function include the processor/process number to which the message being sent and the number of bytes in the message. The arguments of the `nread()` function include the processor/process number from which the message being received and the number of bytes in the message.

In order to make the tracing scheme easy to use, a utility is provided that automatically replaces all calls to `nwrite()` and `nread()` (in the user's parallel program) with instrumented versions, which are named `twrite()` and `tread()`. The utility also inserts the source associated with these instrumented functions by inserting "include" statements in the source. The `twrite()` and `tread()` functions use the task number instead of the processor/process number for identifying the destinations and sources of messages. This allows the user to view communication in terms of logical intertask communication instead of communication between physical processors.

The function `twrite()` first records that communication has been requested to the destination task number, then converts the destination task number (which is passed as an argument of `twrite()`) to a processor/process number and calls `nwrite()` using this processor/process number. The conversion between task number and processor/process number is uniquely defined by the mapfile. The function `tread()` simply converts the task number (passed as an argument of `tread()`) to a processor/process number and calls `nread()`.

In the SPMD mode of programming on the nCUBE, a single program is mapped onto every processor. Functionally distinct tasks can still be executed among the processors in SPMD mode by conditioning the execution of code segments (i.e., tasks) of the single program according to the IDs of the processors. Assume the first instruction in the program is the assignment of the processor ID to a local variable, say `pid`, which can be determined from a function called `whoami()`. Once the processor ID is stored in the local variable `pid`,

then case statements can be used to partition the execution of the program into segments, where each segment is executed by a particular (set of) processor(s). The communication constructs for sending messages (and tracing) in SPMD mode are similar to those described earlier for the MIMD mode.

3.2.3 The Mapping Subsystem

The mapping subsystem in OMARS currently allows the user to select either the initial mapping (used during generation of the tracefile) or mappings produced by one of three mapping algorithms. The currently implemented mapping algorithms are for architectures with hypercube-based interconnection networks. These three mapping algorithms are called the default mapper, the hypersphere mapper, and the greedy search mapper.

The default mapper is a straightforward mapper that does not utilize the intertask communication pattern in deciding the mapping. This algorithm simply maps task number i onto processor number i mod N, where N is the number of processors. The default mapper provides a very simple solution to the mapping problem that can be a useful baseline for comparing other more sophisticated mappers.

The hypersphere mapper [1] employs a nonlinear programming approach for solving the hypercube embedding problem. The basic idea of the approach is to approximate the discrete space of an n-dimensional hypercube, i.e., $\{z : z \in \{0,1\}^n\}$, with the continuous space of an n-dimensional hypersphere, i.e., $\{x : x \in \Re^n \ \& \ ||x||^2 = 1\}$. The mapping problem is initially solved in the continuous domain by employing the gradient projection technique to a continuously differentiable objective function. The optimal task "locations" from the solution of the continuous hypersphere mapping problem are then discretized onto the n-dimensional hypercube. The hypersphere approach can directly solve the problem of mapping T tasks onto N nodes for the general case where $T > N$. In contrast, many of the competing embedding heuristics from the literature only produce one-to-one mappings and can only be directly applied when $T < N$. Refer to [1] for a more detailed description and for performance evaluation studies of hypersphere mapper.

The greedy search algorithm [8] maps the set of tasks onto the set of processors by considering tasks one at a time. To determine which task to consider next, the algorithm selects the task that has a maximum number of neighbors from the task graph already mapped. The algorithm then assigns the task to the next processor, where the processors are traversed in gray-code order. This heuristic is relatively fast, however, its performance in terms of typical objective function values is substantially lower than other more sophisticated approaches.

Other mapping algorithms that will be implemented in future versions of OMARS are

the local search mapper, the mincut bipartitioning mapper, and the simulated annealing mapper. Descriptions of these algorithms are contained in [9] and the references therein.

3.2.4 The Routing Subsystem

The routing subsystem in OMARS currently has two processor-to-processor routing schemes implemented; both of these schemes assume hypercube-based architectures. These routers are called default router and modified multiflo router. Like the mapping subsystem, the routing subsystem provides a "plug-and-fit" framework for implementing and evaluating new and existing routing algorithms. The input to the routing algorithms is the interprocessor communication traffic. These interprocessor demands can only be determined after the task-to-processor mapping is defined. Thus, the output of the mapping subsystem, which is a task-to-processor mapping, is used with the intertask communication pattern (from the tracefile) to determine the interprocessor traffic. Given the interprocessor traffic demands, the routing algorithms determine the paths through the interconnection network to be used for making the required processor-to-processor connections.

The default router is the routing scheme that is commonly implemented in commercial hypercube systems. This is a static routing scheme, which is independent of the assumed interprocessor traffic pattern. To describe the default routing scheme, the topology of the hypercube is first reviewed. In an n-dimensional hypercube, the processors are numbered from 0 to $2^n - 1$ using n-bit binary labels. Two processors in a hypercube are directly connected with a communication link if and only if their binary addresses differ in exactly one bit position. For instance, in a 3-dimensional hypercube processor 2 (010) is directly connected to processors 3 (011), 0 (000), and 6 (110).

To illustrate the default path connecting two processors, consider the generic address labels $S = s_{n-1} \cdots s_0$ and $D = d_{n-1} \cdots d_0$, which represent the addresses of a source and destination, respectively. Assume S and D differ in exactly $1 \leq k \leq n$ bit positions and let b_i, $i = 1, \ldots k$, denote the ith lowest bit position where the labels S and D differ. The first link in the default path from S to D is the direct link that connects $S = s_{n-1} \cdots s_{b_1} s_{b_1-1} \cdots s_0 = s_{n-1} \cdots s_{b_1} d_{b_1-1} \cdots d_0$ to $s_{n-1} \cdots d_{b_1} d_{b_1-1} \cdots d_0$. In general, the ith link in the path is the link that connects processor $s_{n-1} \cdots s_{b_i} d_{b_i-1} \cdots d_0$ to $s_{n-1} \cdots d_{b_i} d_{b_i-1} \cdots d_0$. Thus, at each processor along the path, a single bit of the source address is "corrected," until the path terminates at the destination. As an example, the path connecting processor 0 (000) to processor 5 (101) consists of the direct link from processor 0 to processor 1 and the direct link from processor 1 to processor 5. For further information on default routing and how it is commonly implemented, the reader is directed to [16, 20].

The underlying objective of an optimal routing algorithm is to determine a set of paths,

one for each pair of communicating processors, so that some measure of network contention is minimized. The optimal routing algorithm implemented in OMARS is based on a multi-commodity network flow algorithm called multiflo [4]. Multiflo uses a path flow formulation and a gradient projection method to minimize an objective function of the form

$$D(F) = \sum_{(i,j)\in\mathcal{L}} \frac{F_{ij}}{C - F_{ij}},$$

where $\mathcal{L}$ denotes the set of all network links, C is a constant, and the variable F_{ij} denotes the "flow" on link (i,j). The theoretical foundation of the particular objective function given in the equation above is rooted in queuing theory, and the interested reader is referred to [2] for a more detailed development and explanation.

For the purposes of explanation here, and to provide some intuitive perspective, note that each term in the sum of the objective function is a nonlinear and increasing function of F_{ij}, for $0 \leq F_{ij} < C$. Realizing that F_{ij} is a measure of the traffic that is routed through link (i,j), note that as a particular F_{ij} is increased, the corresponding term in the objective function sharply rises. The values of F_{ij} depend on how the routes between communicating processors are selected. In particular, the value of F_{ij} is equal to the sum of all path flows that utilize link (i,j). So, any candidate technique for solving the optimal routing problem, which is to determine paths for communicating processors so the above objective function is minimized, should tend to select paths so that the values of the link flows (i.e., F_{ij}'s) are relatively uniform. A routing solution where even one link flow value is close to the value of the constant C causes a sharp increase in the value of the objective function.

Routing solutions produced by the multiflo algorithm of [4] generally utilize more than one path to route the traffic between each pair of communicating processors. That is, the algorithm is formulated to allow the total demand between each pair of communicating processors to be split and routed along several distinct paths that connect the source and destination processor. By allowing a continuous splitting of source-destination demand among distinct paths, the routing provided by the multiflo algorithm is provably guaranteed to be optimal (in the sense of achieving a minimum value of the objective function). For practical reasons, the optimal routing solution from the multiflo algorithm is modified by OMARS so only one path is used for each pair of communicating processors. This is accomplished by simply selecting the single path, for each communicating pair of processors, that has the maximum amount of demand routed through it. In practice, the solutions provided by multiflo typically do not have more than three or four "active" paths for each pair of communicating processors. Through experimentation, it has been observed that while the modified version (i.e., single path approximation) of the multiflo routing solution generally

has a corresponding objective value function that is larger than the optimal value, its value is typically much smaller than that associated with the default routing.

3.2.5 The Graphical Displays

Two important objectives that influenced the implementation of OMARS were: (1) to make it easy to use and (2) to present the results from the mapping and routing algorithms in an intuitive and easy to understand manner. These objectives were achieved by making extensive use of graphical displays. The interactive portions of OMARS are completely menu-driven and the user is typically only required to "point and click" the mouse to make selections from the menu items. All of the graphical displays are generated using X-windows system software. The various interactive menus and displays associated with OMARS are highlighted in Section III by "walking through" an example user session.

3.2.6 The Taskfile Generator

The taskfile generator is essentially an output file generator for OMARS. A taskfile contains the mapping and routing information associated with a selected architecture window. The taskfile enables the user to implement the mapping and routing information provided by OMARS on the target parallel machine. Recall that an architecture window is generated each time the user specifies a mapping/routing combination from the mapping and routing subsystem menus. To generate a taskfile, the user selects the "Generate Taskfile" item from the main menu and then selects an architecture window using the mouse. The taskfile is formatted in a way that makes it easy to implement the mapping and routing information on the assumed parallel platform. Current taskfiles are formatted to be used with the nCUBE 2, although formats are planned for Intel's i860 and Paragon machines.

3.3 Example User Session of OMARS

The objective of this subsection is to walk through a typical user session of the OMARS tool as a means of illustrating its "ease of use" and to demonstrate the potential advantage of using the mapping and routing information provided by the tool. Because of space limitation, the process of instrumenting the parallel code for the purposes of tracing is not included here. The displays for the session are based on a tracefile named `trace.16`, which was generated by tracing the execution of a SPMD program executing on a 4-dimensional subcube of the nCUBE 2.

To start the example user session, the command `omars trace.16` was executed, which generated the two windows shown in Figure 9. The window on the left side of the figure

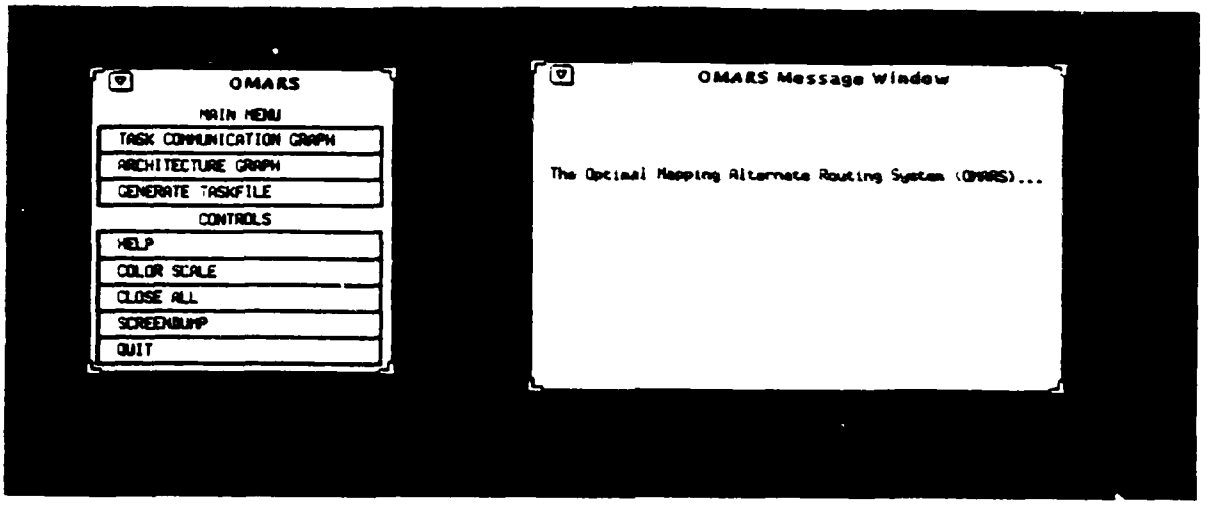

Figure 9: Initial windows generated by executing OMARS.

has two parts: the main menu and the controls; the window on the right side is called the message window. The message window is used by OMARS to convey useful information and/or instructions at various stages of interaction with the user.

The functions associated with the three buttons within the main menu will now be illustrated. By selecting the task communication graph button, the intertask communication pattern among the tasks is displayed graphically, as shown in Figure 10. When the task communication graph is initially displayed, only the nodes of the graph are drawn (one for each task). To determine which tasks a particular task sends messages to, the user selects the desired origin task and OMARS draws arcs from this origin task to all associated destination tasks. Shown in Figure 10 is the result of the user selecting tasks 3, 6, and 12. The color of each arc corresponds to the number of messages transmitted to the associated destination. The color scale, which is displayed on the upper left side of the window, is shown in gray-scale here. To erase all arcs in the task communication graph, the user can select the clear button located above the color scale. To display all arcs of a particular color (i.e., traffic level), the user can press the associated button of the color scale and all arcs of that color will be displayed.

By selecting the architecture graph button, a menu of mapping choices is generated as shown in Figure 11. After selecting a mapping algorithm from the mapping menu, a menu of routing choices is displayed, as shown in Figure 12. For each mapping/routing combination selected by the user, an architecture graph window is generated. Figure 13 shows the result of selecting four distinct combinations of mapping and routing choices: (1) greedy search mapping with default routing, (2) greedy search mapping with modified multiflo routing, (3) hypersphere mapping with default routing, and (4) hypersphere mapping with modified multiflo routing. The nodes of an architecture graph represent the processing elements (PEs)

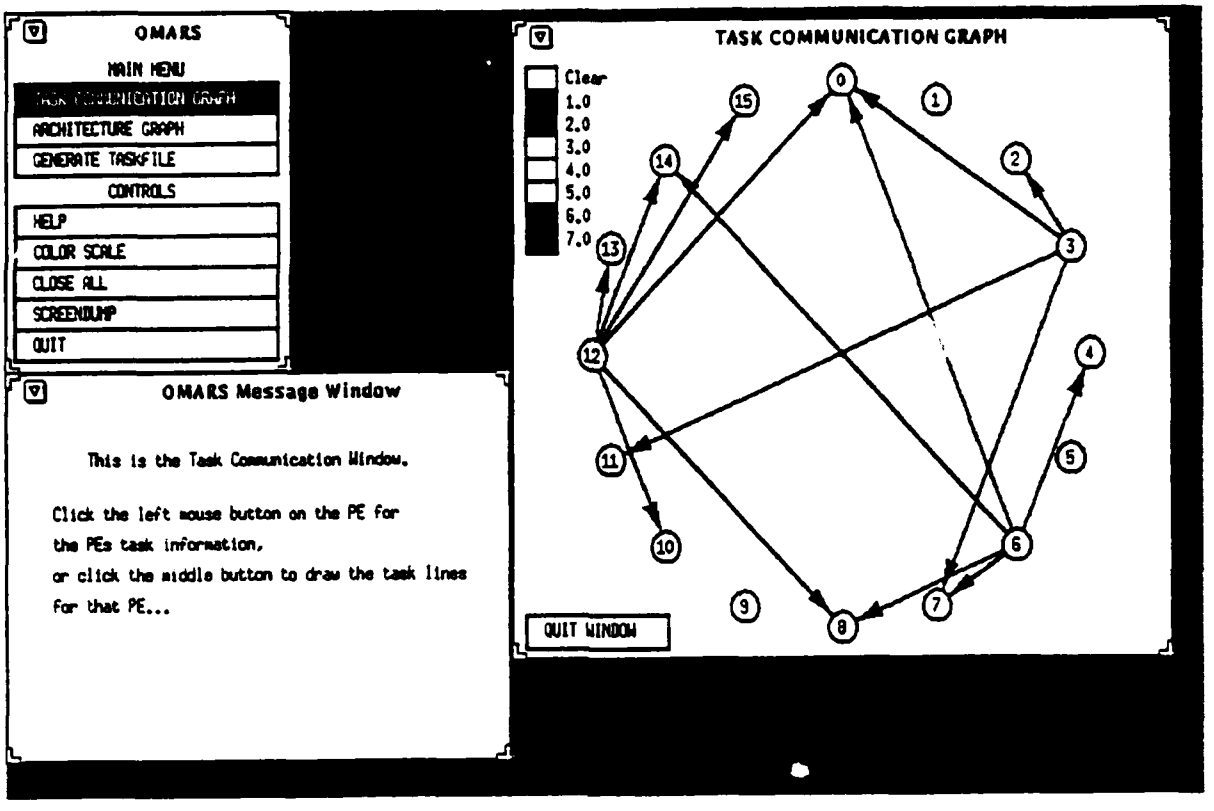

Figure 10: The result of selecting the task communication graph button.

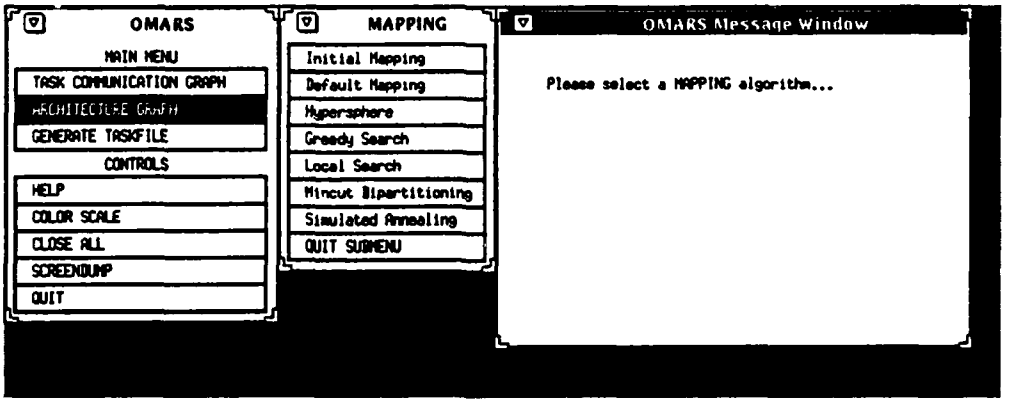

Figure 11: After selecting the architecture graph button, the user is prompted to select a mapping algorithm from the mapping menu.

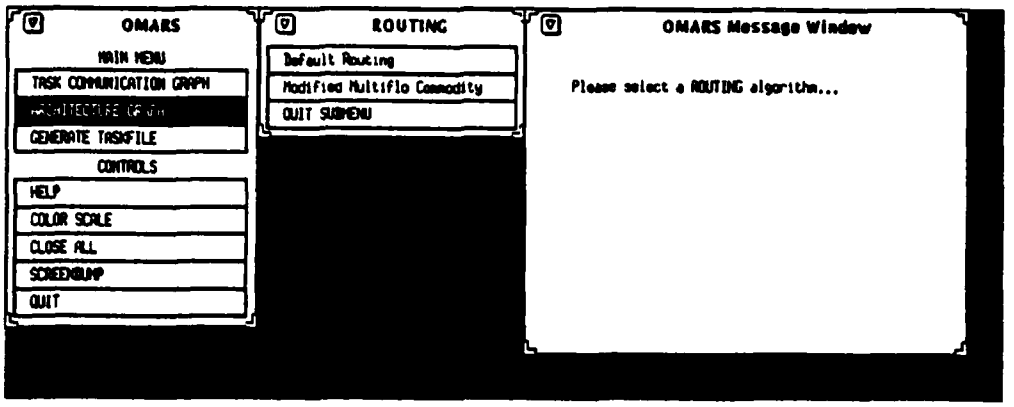

Figure 12: After selecting the architecture graph button and selecting a mapping algorithm from the mapping menu, the user is prompted to select a routing algorithm from the routing menu.

of the parallel architecture and the edges represent the interconnecting communication links. The color of each communication link corresponds to the link usage factor, which is the number of message that use the link. The nodes are colored based on the number of tasks mapped to each PE. Also displayed in each architecture window is some textual information that includes the name of the tracefile used and some numerical metrics.

The numerical metric labeled PE load variance indicates the variance in the number of tasks mapped to each PE. If the same number of tasks are mapped to each PE, then the value of this metric is zero, otherwise it is a positive number; larger values imply more variation in the number of tasks mapped to each node. The inter-task communication metric is the total number of messages sent between all tasks. The network traffic is the total number of message that are sent through the interconnection network. The average distance is the average number of links used to establish paths for all communicating tasks. Most of the mapping algorithms are based on minimizing this metric. The maximum link usage is the maximum number of messages sent across any communication link. Optimal routing techniques will typically decrease the value of this metric.

From the four architecture graphs shown in Figure 13, the choice of hypersphere mapping with modified multiflo routing has the best overall metric values. To generate a taskfile for this mapping/routing combination, the user selects the generate taskfile button, which causes a pop-up window to be generated prompting the user to select an architecture window. The user then selects any portion of the desired architecture window, and another pop-up window is generated that includes a proposed name for the taskfile to be generated, as shown in Figure 14. At this point, the user can either accept the proposed taskfile name, type another name,

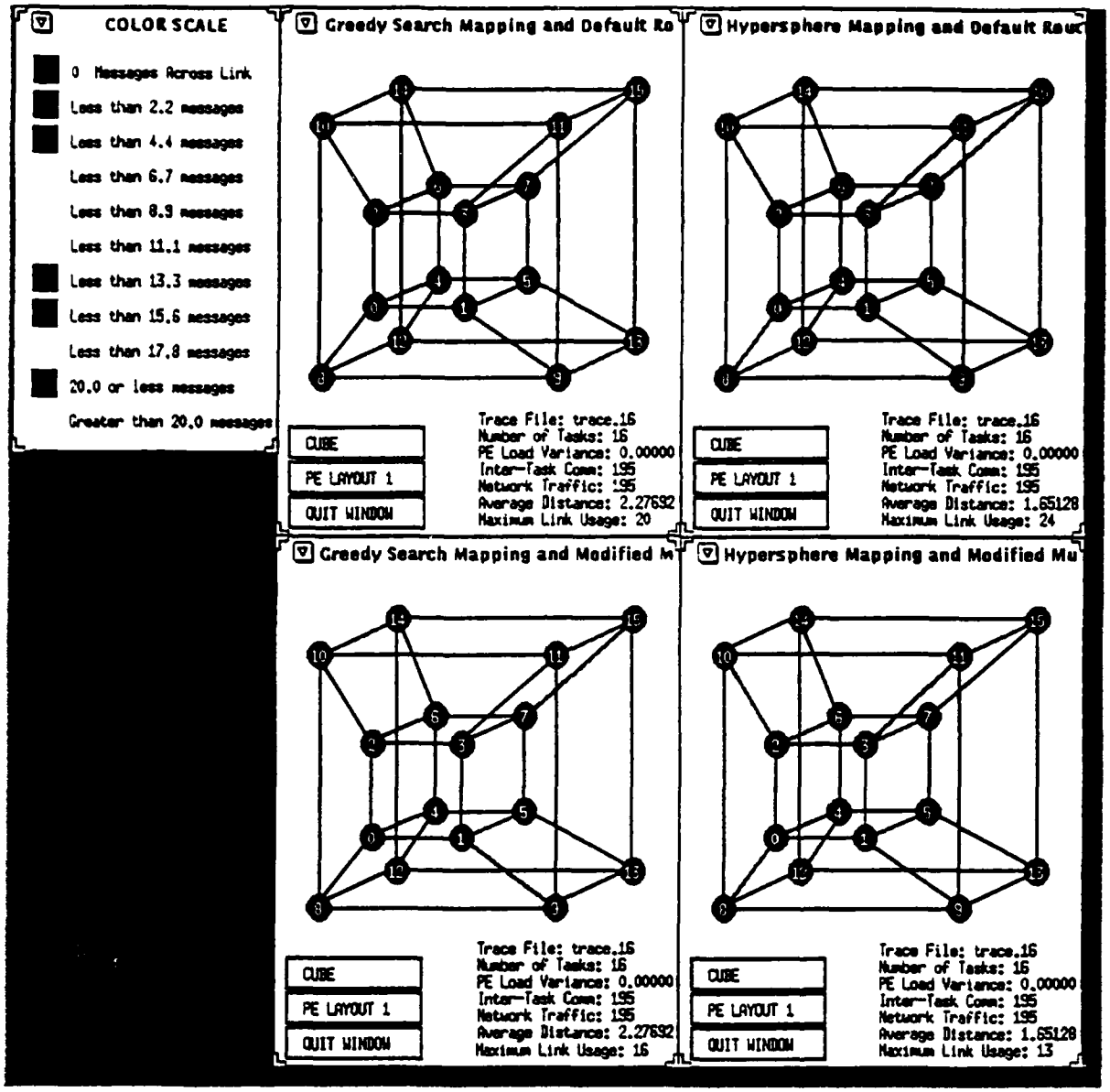

Figure 13: After selecting the architecture graph button and selecting a mapping algorithm and a routing algorithm, an associated architecture graph is displayed. Shown in the figure are four distinct architecture windows associated with various mapping/routing combinations.

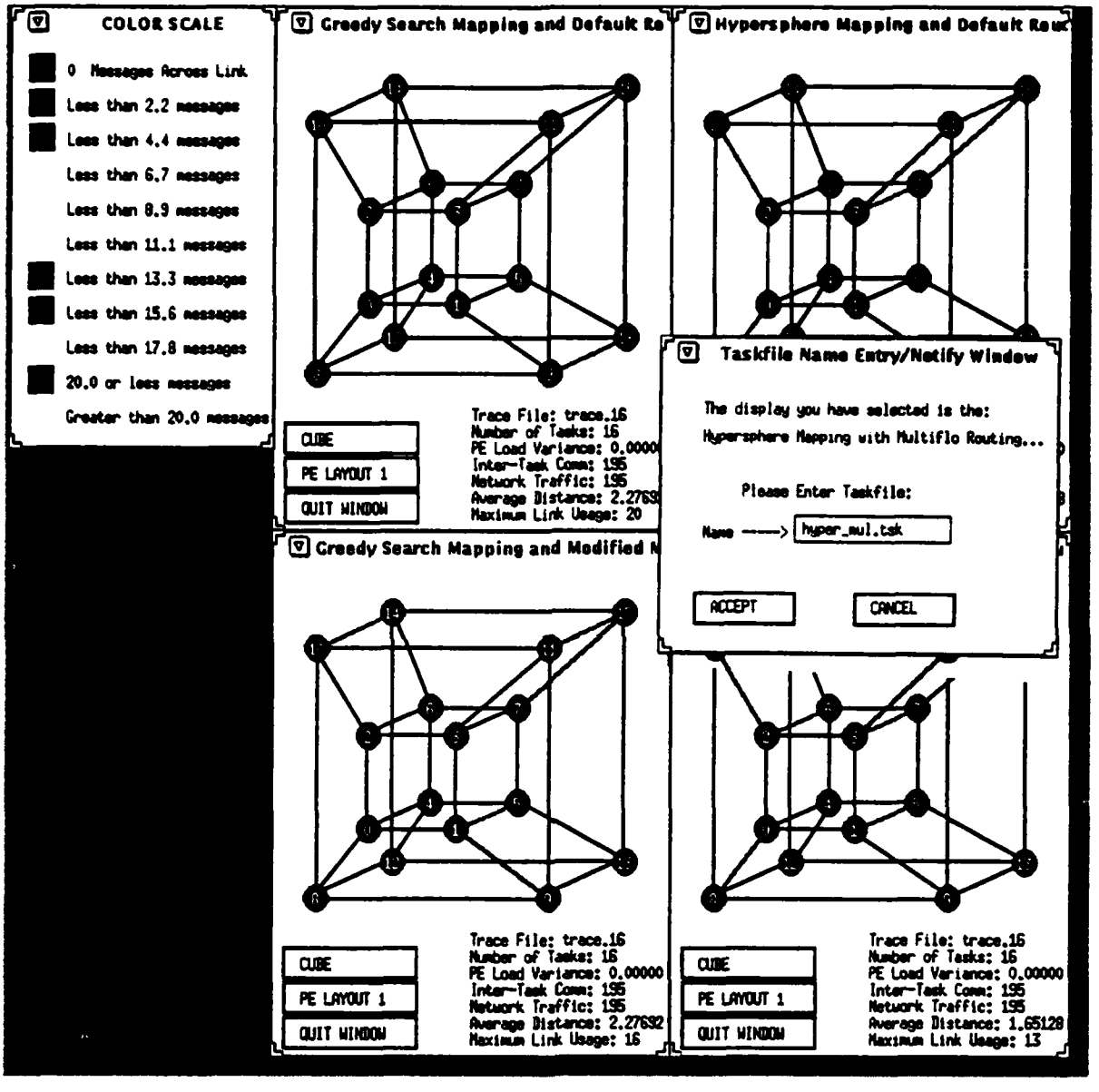

Figure 14: To generate a taskfile, the user selects the generate taskfile button, then selects a desired architecture window. OMARS then prompts the user to name the associated taskfile to be created.

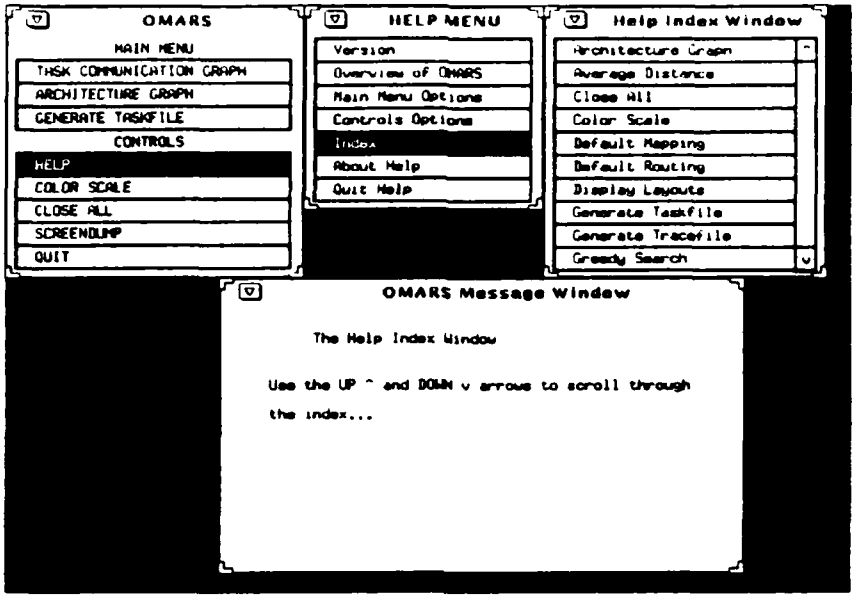

Figure 15: Selecting the help button generates a help menu to aid the user in understanding various aspects of OMARS. Shown in the figure is the result of selecting the index button from within the help menu.

or cancel the generation of a taskfile.

An important feature of OMARS is that on-line help is implemented. From the control menu the user can select the help button, which generates a help menu as shown in Figure 15. In this figure, the index button within the help menu was selected, which generated a help index window. The help index window contains definitions and descriptions of virtually every aspect of OMARS.

Because of space limitation, detailed performance evaluations associated with incorporating taskfiles generated by OMARS for nCUBE programs are not included here. A general trend, however, is that execution time tends to be strongly correlated (typically) with the value of the maximum link usage factor. This is not surprising as the switching methodology used by the nCUBE 2 is essentially a variant of virtual cut-through, refer to [20] for more details.

3.4 Configuration Management for OMARS

The software development for OMARS took place at two different locations. The graphical displays and user interfaces for the tool were developed primarily at Rome Laboratory, Rome, New York. The data structure development and algorithm research and implementation was done primarily at Purdue University, West Lafayette, Indiana. In this distributed software development environment, the importance of having systematic procedures and guidelines for version control and configuration management were realized early-on.

A configuration management process was implemented using the UNIX SCCS (source code control system) tool. A description of the configuration management process used to develop software for OMARS is outlined in this subsection.

3.4.1 Responsibilities of the Software Developers

Individual software developers were able to electronically check items in and out of a central SCCS library, which was stored on a hard-disk at Rome Laboratory. Each software developer was required to adhere to the following guidelines and responsibilities.

- Use only those commands provided by the configuration manager (via linking to an alias file) when accessing files kept in the SCCS library.
- Provide comments upon returning edited files to the library. These comments are incorporated in change records kept for each file in the library.
- Add a standard header to new files that are created for the library.
- Submit change proposals for the system Makefile. (This was implemented by sending proposed changes via email to the configuration manager.)

3.4.2 Initial Library Files

To give an indication of the number and types of files managed by the SCCS library, listed below are the files associated with version 1.1 (initial version) of the OMARS tool.

HEADER FILES

defaults.h	defs.h	globaldef.h	trace_spmd.h
defines.h	global.h	trace.h	

C FILES

bitinv.c	draw_init.c	multi_out_to_link.c
button.c	eval_cost1.c	net_file_gen.c
call_mappers.c	eval_grad1.c	norm_2.c
call_routers.c	expose.c	r_spread_x_out.c
compute_avg_dist.c	gen_trace.c	readtrace.c
compute_max_util.c	greedy_mapper.c	route_parse.c
compute_net_traffic.c	hypersphere_mapper_g.c	sim_anneal_mapper.c
compute_variance.c	init_positions.c	table_output.c
convert.c	local_search_mapper.c	trace.c
convert_to_PE_odpairs.c	main.c	trace_mimd.c
default_mapper.c	map_x_onto_hypercube.c	traf_file_gen.c
draw.c	mincut_bipart_mapper.c	

FORTRAN FILES

dcal.f	driver.f	prsumm.f	shorpape.f
delay.f	load.f	round_multiflo.f	

OTHER FILES

Makefile	netwrk.prm	paths.blk	trace_file
convrg.prm	oconfig.tex	round_comp	
hyper_default	param.dim	round_multi	

3.4.3 Commands for Accessing the SCCS Library

Each user set up a personal working directory in his/her local workstation for the purpose of software development. Software developers used the following commands, provided by the configuration manager, in order to check out, edit, test, debug, check in, and check the status of items kept in the SCCS library.

oclean : removes all files from the current directory that can be retrieved from the library. It does not remove any files currently checked out for editing.

ocmds : gives a summary of the OMARS library commands.

ocreate *filename* : creates a new library file by renaming *filename* to *,filename*, places a copy of *filename* into the library, and gets a copy of *filename* from the library.

odone *filename* : checks in a file that has been checked out for editing and prompts the user for comments, which are placed in the history for documenting changes to *filename*.

oedit *filename* : checks out a file for editing, and locks it against concurrent changes.

oget *filename* : gets a read-only copy of a file from the library; changes to this file are not tracked because it is not checked out to be edited.

ohelp *errno* : gives detailed information about error codes returned by the SCCS tool.

orpt *filename* : reports on the history of *filename*.

ostat : lists the files being edited, the old and new version numbers, the user(s) that checked out the file(s), and the time and date the file(s) was (were) checked out.

ounedit *filename* : returns *filename* to its previous condition by removing all changes made to *filename* since it was checked out.

3.4.4 Benefits of the Configuration Management Process

The configuration management process allowed multiple software developers to simultaneously design, code, and test new functionalities for OMARS with very little contention and/or confusion. Because past versions for all files were saved and documented within SCCS, software developers were able to roll back to various combinations of file versions for the purposes of recreating executables from the past. The history kept for each file was especially useful in tracking changes and finding bugs. Perhaps the most important aspect of the process was that each file could only be checked out (for the purposes of editing) to one person at a time. This serialization eliminated the possibility that two programmers might, unknowingly, attempt to make modifications to the same file at the same time.

3.5 Conclusions and Future Work

The development of OMARS is an ongoing effort. The current version of OMARS is about 12,500 lines of code (about 11,000 lines in C and the remaining in FORTRAN). This does not include the modifications and additions (for alternate routing) that have been made on the OS source code for the nCUBE 2, which is written in assembly language.

The primary goal of OMARS is to provide software engineers with some tools for addressing relatively low-level issues (i.e., mapping and routing) that arise when developing large software applications on parallel processing platforms. Future work will include the implementation of other mapping and routing schemes for various types of architectures. Plans are currently being made to introduce OMARS, on an experimental basis, to students in a parallel programming class; as a means of getting user feedback. Plans beyond this initial release are to eventually make the tool available to the general public.

Special Note Regarding Figures in Section 3

The graphical displays for OMARS are implemented in color for viewing on a color monitor. The screen dumps from OMARS shown in this section are in black & white, and thus the effect of having color for the arcs, nodes, and color scale is not shown. The color component is an especially important factor for the displays associated with Figures 10, 13, and 14.

4 Future Research and Development Directions

The following is a list of directions/tasks that we would like to pursue in the future.

- completion of alternate routing implementation for OMARS
- OMARS display research and development
  - develop new aggregate and/or more user friendly metrics
  - color scale experiments
  - grey scale experiments
  - average distance normalization
  - average distance display
  - network traffic normalization
  - network traffic display
  - maximum link utilization normalization
  - maximum link utilization display
  - PE load variance normalization
  - PE load variance display
  - 32 PE display
  - 64 PE display
- basic mapping research
  - minimize maximum link utilization
  - for mesh interconnection networks
- experiments using OMARS output on simulated program behaviors
- OMARS with real applications

References

[1] J. K. Antonio and R. C. Metzger, "Hypersphere mapper: a nonlinear programming approach to the hypercube embedding problem," *Proc. 7th Int'l Parallel Processing Symposium*, Apr. 1993, pp. 538–547.

[2] J. K. Antonio, W. K. Tsai, and G. M. Huang, "Time complexity of a path formulated optimal routing algorithm," *IEEE Trans. Automatic Control*, scheduled for publication, Dec. 1993.

[3] M. S. Bazaraa and C. M. Shetty, *Nonlinear Programming: Theory and Applications*, John Wiley & Sons, NY, NY, 1979.

[4] D. P. Bertsekas, B. Gendron, and W. K. Tsai, "Implementation of an optimal multi-commodity network flow algorithm based on gradient projection and a path flow formulation," MIT, Cambridge, MA, LIDS Rep. No. LIDS-P-1364, Feb. 1984.

[5] S. Bettayeb, Z. Miller, and I. Sudborough, "Embedding grids into hypercubes," *VLSI Algorithms and Architectures: 3rd Aegean Workshop on Computing*, Lecture Notes in *Computer Science*, Springer Verlag, **319**, pp. 201–211, 1988.

[6] K. Bhat, "On the complexity of testing a graph for N-cube," *Information Processing Letters*, **11**, pp. 16–19, 1980.

[7] L. N. Bhuyan, "Interconnection networks for parallel and distributed processing," *Computer*, vol. 20, no. 6, June 1987, pp. 9–12.

[8] W.-K. Chen and E. Gehringer, "A graph oriented mapping strategy for a hypercube," *Proc. Third Conference on Hypercube Concurrent Computers and Applications*, 1988, pp. 200–209.

[9] W. K. Chen, M. F. M. Stallman, and E. F. Gehringer, "Hypercube embedding heuristics: an evaluation," *Int'l J. Parallel Programming*, vol. 18, no. 6, 1989, pp. 505–549.

[10] G. Cybenko, D. Krumme, and K. Venkataraman, "Fixed hypercube embedding," *Information Processing Letters*, **25**, 1987, pp. 35–39.

[11] M. J. Flynn, "Very high-speed computer systems," *Proceedings of the IEEE*, vol. 54, no. 12, Dec. 1966, pp. 1901–1909.

[12] G. Fox, "A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube," *The IMA Volumes in Mathematics and its Applications, Volume 13*, Martin Schults, editor, Springer Verlag, 1988.

[13] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley, "PICL: a portable instrumented communication library, C reference manual," Technical Report ORNL/TM-11130, Oak Ridge National Laboratory, Oak Ridge, TN, July 1990.

[14] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley, "A user's guide to PICL, a portable instrumented communication library," Technical Report ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, TN, Oct. 1990.

[15] M. T. Heath and J. A. Etheridge, "Visualizing the performance of parallel programs," *IEEE Software*, vol. 8, no. 5, Sept. 1991.

[16] K. Hwang, *Computer Architecture and Parallel Processing*, McGraw-Hill, New York, NY, 1984, pp. 339–340.

[17] P. Kermani and L. Kleinrock, "A tradeoff study of switching systems in computer communications networks," *IEEE Trans. Computers*, vol. C-29, no. 12, Dec. 1980, pp. 1052–1060.

[18] S. Kirkpatrick, C. Gelatt, Jr., and M. Vecchi, "Optimization by simulated annealing," *Science*, pp. 671–680, 1983.

[19] R. C. Metzger and J. K. Antonio, "Research issues for executing real-time C3 applications on parallel processing systems," *Proc. IEEE Workshop on Real-Time Applications*, May 1993, pp. 81–86.

[20] nCUBE Corporation, *nCUBE 2 Processor Manual*, Order # 101636, nCUBE Corporation, Dec. 1990.

[21] H. J. Siegel, J. K. Antonio, and K. J. Liszka "Metrics for metrics: why it is difficult to compare interconnection networks OR how would you compare an alligator to an armadillo?," *Proc. The New Frontiers: A Workshop on Future Directions of Massively Parallel Processing*, Oct. 1992.

[22] A. Wagner, "Embedding arbitrary binary trees in a hypercube," *Journal of Parallel and Distributed Computing*, **7**, pp. 503–520, 1989.

[23] A. Wu, "Embedding of tree networks into hypercubes," *International Journal of Parallel and Distributed Computing*, **2**, pp. 238–249, 1985.

*U.S. GOVERNMENT PRINTING OFFICE: 1994-510-117-50011

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.